Fundamentals of Firewalls and Packet Filtering

Kramse
hlk@zencurity.com

WARNING: This is currently a very rough draft! Feedback welcome.

Feedback very welcome!

Release 1.0-DRAFT a version produced for PROSA courses in August 2025.

Contents

Abstract

Networks are increasingly complex and we need control over chaos. This requires us to consider confidentiality, integrity and availability - while working efficiently, distributed and freely. The task seems impossible.

Since the 1990s we have used firewalls to enhance network security, but what are firewalls really doing. Firewalls are network security devices which can inspect traffic and allow or disallow flows to pass through them.

What firewalls and similar functions can do is to reduce the amount of security threats hitting our systems, reduce the likelihood that malware can communicate back to the control systems – and often the firewall log can tell us what happened during a breach of security.

This book aim to present the concept of firewalls and packet filtering allowing newcomers to learn what network filtering devices can do, and what the limits are. Goal is to teach principles and less about technology. The book contains complete examples using OpenBSD Packet Filtering (PF) syntax, and reference a few other network filtering systems.

Keywords: Firewalls, networking, Internet security, network segmentation, netflow, monitoring.

Preface

Welcome to the book project *Fundamentals of Firewalls and Packet Filtering* published by Henrik Kramselund, Zencurity Aps.

This book is meant as an introduction about firewalls to be read before diving into vendor documents for commercial or open source firewalls. So the main goal is to be a generic firewall book along the lines of the classic text *Firewalls and internet security; repelling the wily hacker. Second edition* by Cheswick, Bellovin, and Rubin (2003) - even if this may never be as detailed.

Chapter 1

Introduction

1.1 Introduction and Goals

The Internet and connected networks are increasingly complex and we need security. Since the 1990s we have had technologies which we today categorize as *firewalls*. Along the way these have expanded in many directions, so today it is hard to understand, *What is a firewall* To make this more approachable we will start from the bottom and discuss filtering of network traffic.

The shortest answer to what is a firewall might be, some internet filtering device, a combination of hardware and software. This answer requires the reader to know more than just *firewalls*, so this book will try to point to other resources along the way. We also want this book to describe the features in a modern firewall by starting at the most basic building blocks and components.

So in essence this book cover mostly low level network filtering security mechanisms for protecting networks, systems and data.

The book has multiple goals

- Be short and introductory – enable further exploration into filtering traffic afterwards
- Be as vendor neutral as possible – leave out many details about products
- Introduce the firewall concept including the notion that a firewall often is multiple devices working together
- Be practical – allow you to actually go and get started by configuring some filtering technologies
- Recommend a holistic strategy which is considered long term beneficial

1.2 Intended Audience and Content

The intended audience for this book are people that have heard the term firewall and wants an introduction to this concept. This book presents the technology and certain terms for a foundational understanding of network filtering for newcomers to the niche field.

Seasoned firewall administrators may find a hint or two, but will mostly likely not learn anything new.

While this book may not teach you how to write complete security policies, we have some tips for writing firewall policies with default deny and managing them.

We also have devices and systems being used all the way to application level that includes the term firewall, for example a web application firewall (WAF) which filters HTTP/HTTPS requests before reaching application servers. We will not cover application level proxy type systems.

We will only cover parts of network segmentation in this book, but consider this a whole subject in itself, and orthogonal to firewalls and filtering. Firewalls and filters control traffic flow between network segments, network segmentation enable filtering by splitting networks depending on security requirements.

We highly recommend network segmentation and adding filtering policies to your segmented networks.

The book is divided into the following chapters:

- Chapter 2 Introduce a few security principles that should guide you
- Chapter 3 Short listing of some common security problems in networks

- Chapter 4 Explain what we consider a firewall
- Chapter 5 Design and Planning of Firewalls
- Chapter 6 Creating firewall policies
- Chapter 7 Detail in depth some of the core firewall features
- Chapter 8 Describe stateless and stateful filtering, goals and differences
- Chapter 9 Introduce protection of single systems with firewalls
- Chapter 10 Introduce a DMZ and how to protect network segments
- Chapter 11 A complete example is shown that can be used for protecting a small network

1.3 Vendor Neutral

This book is about firewall principles and less about specific firewall technologies from certain vendors.

So a clear goal for this book is to be vendor neutral. Throughout the book we will show small practical examples of the theory applied using common devices. Specifically the book will use a few larger firewall examples using OpenBSD PF (Packet Filter). This firewall technology is BSD licensed open source and has been ported to other operating systems like FreeBSD and NetBSD, but also Mac OS X include this technology.

Wikipedia has a list of examples where the technology is used https://en.wikipedia.org/wiki/PF_(firewall)

Most features can be found in similar products from other providers, and we urge you to do as much as possible with your existing devices.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic Indicates new terms, URLs, directory names, and filenames.

`Constant width bold` Shows commands or other text that should be typed literally by the user. Also used for code and commands, as well as within paragraphs to refer to command-line tools and their options.

Part I: Firewall Principles

Chapter 2

Core Security Principles

2.1 What is Security

To be able to reason about security we need to introduce what we mean when we say, secure and protected. For that purpose we often use the CIA model describing *confidentiality*, *integrity* and *availability*.

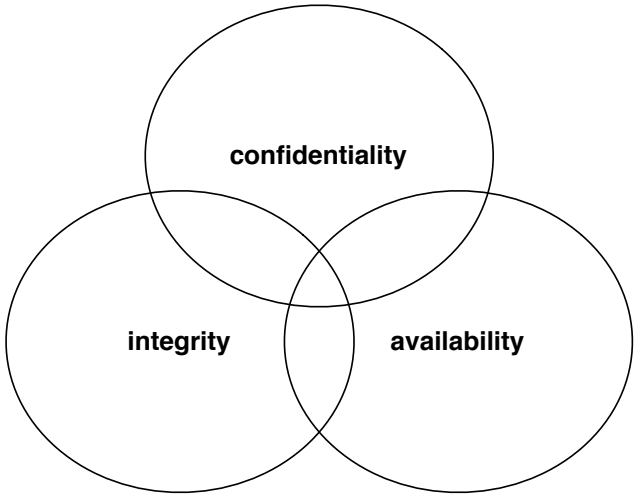


Figure 2.1: The CIA model provides high level guidance for organizations

We want to protect something, and we can use these as guiding principles:

- Confidentiality - data kept away from those unauthorized
- Integrity - data is not subjected to unauthorized changes
- Availability - data and systems are available when needed to the right users

We would typically use these in our daily work to describe problems, vulnerabilities and use them to guide our responses. Some organizations prioritize confidentiality above all else, while other organizations might not have sensitive data and would rather have their systems available all the time.

Along with the above listed we can find models and sources describing privacy along with compliance, laws, regulations – external factors relating to security.

2.2 Goals of Security

Enterprises today have a lot of computing systems supporting the business needs. These are very diverse and often discrete systems which have accumulated over years of use. There is a high degree of complexity and employees have joined and left the organisation. Software we use have bugs and errors, which we will call *vulnerabilities* if they can impact security.

Keeping systems secure requires us to be able to know when we have a security breach, and to avoid them as best as possible. To simplify and sum up we need:

- Prevention - means that an attack will fail, blocked or avoided
- Detection - determine if attack is underway, or has occured - report it
- Recovery - stop attack, assess damage, repair damage, incident response

Prevention can be turning off features, making them unavailable to attackers, with hardening practices. Since this is not always done we can also use network wide preventive measures, such as firewalls.

> **Definition 1-1.** A *security policy* is a statement of what is, and what is not, allowed.

> **Definition 1-2.** A *security mechanism* is a method, tool or procedure for enforcing a security policy.

Definitions borrowed from *Computer security: Art and science, 2nd edition* by Matt Bishop (2019)

Security mechanisms can prevent, detect or help recover, while logging information about what happened and when did it happen.

In this book we will use the words policy to refer to *firewall policies*, which are directly implementing a security policy for technical mechanisms. These should be made with the guidance of security policies for the organization, and implement these upper policies.

2.3 Security Principles

To aid in implementing secure systems we have a number of *security principles* which can be considered design patterns for systems to follow. Below are only 3 examples which are most relevant to the discussion of firewalls. The definitions below are also borrowed from Matt Bishop (2019) but originates in the classic paper *The protection of information in computer systems* by Saltzer and Schroeder (1975) which contain more principles.

Principle of Least Privilege

> **Definition 14-1** The *principle of least privilege* states that a subject should be given only those privileges that it needs in order to complete the task.

We can use this when we consider access to a server. The users, human or other systems, need only access to the service, while management functions should only be available to those who require it.

Principle of Fail-Safe defaults

> **Definition 14-3** The *principle of fail-safe defaults* states that, unless a subject is given explicit access to an object, it should be denied access to that object.

This can applied in network security and some examples are the following:

- Default access *none*
- In firewalls default deny – that which is not allowed is prohibited
- Real world example, OpenSSH config files that come with `PermitRootLogin no`
- Newer devices today can come with no administrative users with default passwords, while older devices often came with default admin/admin users

We will discuss firewall policies multiple times throughout and even a very open policy applied with default deny is better than a default allow policy.

Principle of Economy of Mechanism

> **Definition 14-4** The *principle of economy of mechanism* states that security mechanisms should be as simple as possible.

Within network security some examples might be:

- Simple $->$ fewer complications $->$ fewer security errors
- Use a high quality WPA passphrase for Wi-Fi encryption instead of MAC address based authentication, since WPA encryption is strong and managing MAC addresses for a whole network is complicated

All of the above are intended for helping you choose between different options when implementing network security, including firewalls and/or filtering devices, and configuration of these.

2.4 Defense in Depth

An important realization is that there is no such thing as 100% security. The fact is that security incidents occur, and we cannot expect our protection to always succeed. To ensure we catch most attacks, and do so as quickly as possible we can often use the strategy *defense in depth*.

Defense in depth means that we have multiple security mechanisms working together to provide security. Even if one should suffice we have found that it might fail, and then having multiple lines of defense will allow one to fail, and the others would still keep our systems protected. The likelihood and risk of all our defenses having a weakness and fail at the same time should then be reduced.

To this end we often implement security in multiple places and we select mechanisms that are suitable, with less resource spent we get a higher efficiency. You always have limited resources for protection - use them as best as possible

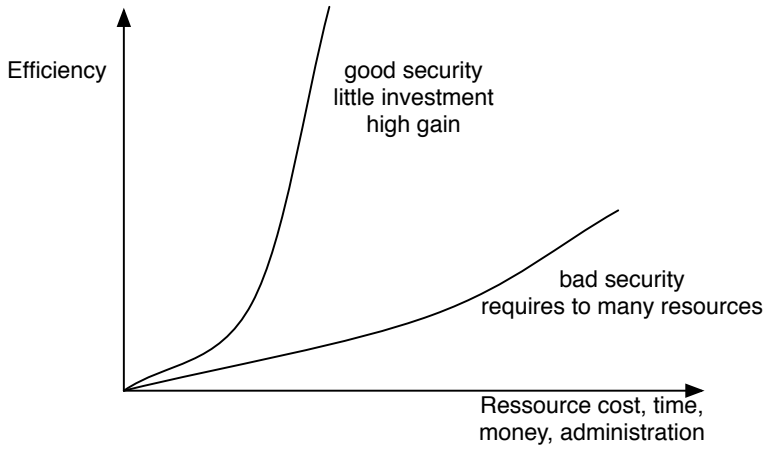


Figure 2.2: Low resource cost and highly efficient are preferred

In this book there will be specific approaches grounded in experience and analysis to describe methods for optimizing traffic filtering. We recommend that you describe alternatives for your own systems, mechanisms and network to find the optimal solutions.

Often you can filter traffic on multiple devices, but depending on the specific traffic patterns and the goal you may see a large difference in resource usage depending on which device that does it. Remember availability is a security parameter too, and an overloaded device is not very useful.

Chapter 3

Security Problems in Networks

3.1 The Internet Protocols TCP/IP

For this book we will only consider networks based on the Internet Protocol (IP) suite, or TCP/IP for short. These are the foundation for the world wide network we call the Internet. These protocols were designed and implemented in the early 1980s and during that time security was not a primary factor. Since then we have seen how important security has become and we aim to reduce security incidents when building networks and connecting them to the Internet.

The most basic use-case is a client connecting to a server, as shown in figure 3.1.

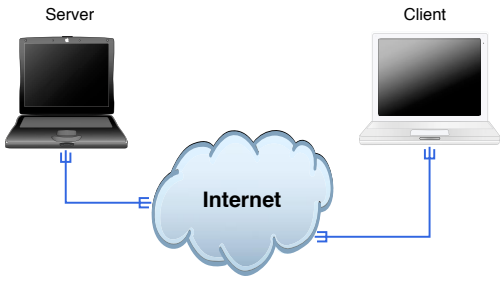


Figure 3.1: Client and server communication

We can also characterize the Internet as having the following properties, among others:

- Clients and server communication with roots in the academic world
- Protocols are old. The canonical document RFC0791 describing the Internet Protocols (IP) is from 1981. The protocols were deployed on the internet around 1983
- Very little in the older protocols were encrypted, luckily most web traffic today is using TLS as in the HTTPS family of protocols
- A common addressing scheme – even though IP version 4 (IPv4) uses 32-bit IP addresses, while IP version 6 (IPv6) uses 128-bit IP addresses
- A single Domain Name System (DNS) a hierarchical and decentralized naming system based on 13 root name servers which are operated by 12 independent organisations[1]
- Simple technologies with best-effort and few requirements, combined and layered to provide services

On a more global scale we should also consider that the Internet is more like an anarchistic patchwork of networks. Therein lie the name internet, from internetworking – a network of networks. These networks are not owned by a single entity, but independent. We have all agreed more or less to use the internet protocols, and abide by some rules.

This result in a picture much like the one shown in figure 3.2.

This show how a user with device on the lower right might connect to servers on the left, using their own internet service provider (ISP), going through multiple networks and internet exchanges before reaching a hosting environment on the left which has the services needed.

[1] see more about the DNS root servers at https://root-servers.org/

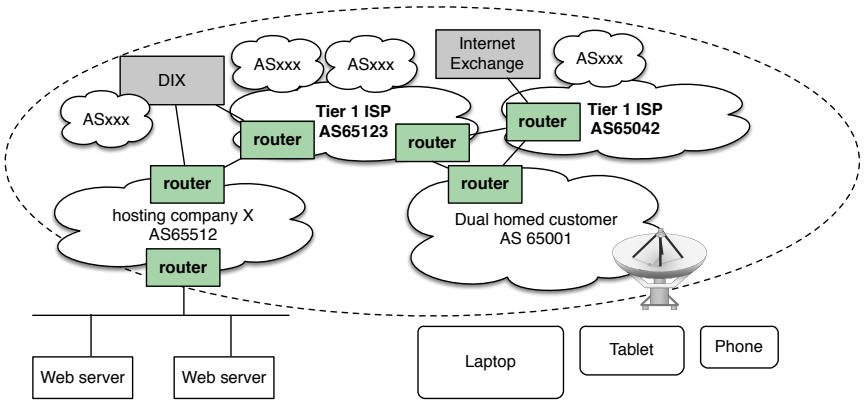


Figure 3.2: The Internet is a network of networks, identified by AS number

This all functions mostly and for most networks, except when there are problems.

3.2 Security Problems in Networks

Network security problems is not something new that appeared in the last few years. It is something that has been around for lots of years. Some of my favourite papers are *Security problems in the TCP/IP protocol suite* by Steven M. Bellovin – published in (1989), and the follow up *A look back at "Security problems in the TCP/IP protocol suite"* from (2004).

These papers list security issues with TCP/IP and the Internet, on different layers.

To name a few of the problems described in the original:

- Routing attacks – IP source routing, Routing Information Protocol (RIP), Internet Control Message Protocol (ICMP) redirect
- Source address spoofing – Address Resolution Protocol (ARP) and IP spoofing
- Authentication attacks – trust between systems based on source IP address
- Sequence number spoofing – Transmission Control Protocol (TCP) sequence numbers could be abuse
- Dangerous defaults such as Simple Network Management Protocol (SNMP) with community string *public*
- Ethernet eavesdropping and host-spoofing, ARP spoofing – intercept, modify, and forward packets
- Problems with Domain Name System (DNS) – paper list only a few problems, more have surfaced over the years

While we have added new technologies there is a high degree of similarity between the problems we faced on the internet before we personally even started to use the Internet, and today which is decades after.

We also see these problems across our network, on all layers, in all corners.

3.3 Network Knowledge Needed

To work efficiently within the network security area it is required to have some network knowledge.This is also recommended before diving into this book and firewalls in general. You may be an Internet user, but do you know packets, connections, protocols and services.

To work with network security the following protocols are the bare minimum to know about.

- IPv4 & IPv6 – the basic packet fields source, destination,
- Address Resolution Protocol (ARP) for IPv4
- Neighbor Discovery Protocol (NDP) for IPv6
- Internet Control Message Protocol – ICMPv4 & ICMPv6
- User Datagram Protocol (UDP) for sending data
- Transmission Control Protocol (TCP) for sending data in flows with error detection
- Dynamic Host Configuration Protocol (DHCP) for assigning IP address to clients
- Domain Name System (DNS) for name lookup

These protocols are part of the Internet Protocol (IP) suite, or TCP/IP for short. The canonical document describing these are the Request for Comments (RFC) series of which the initial IP protocols are described in

RFC0791: *Internet protocol* from (1981).

Currently we do not recommend diving into the RFC documents for learning networks, as more easily accessible resources exist. One such is the book *Practical packet analysis, 3rd edition* by Sanders (2017) – which is updated and describes the most common protocols in enough detail.

Your home network make use of these, so you can learn and discover these at home too.

3.4 Internet Protocols version 6

Since the Internet has grown tremendously there was a need to expand the address space, which was done by introducing Internet Protocols version 6 (IPv6) with longer addresses. The two versions currently co-exist on the networks we build, and and have some overlap within DNS etc.

Note: even though you may have decided not to spend time researching and implementing IPv6 it is probably on your network already. Many devices have IPv6 enabled by default and are actively sending packets around in your network.

There have been examples of hackers using IPv6 to communicate across networks without beeing seen by the administrators, so be warned if you try to ignore IPv6.

My recommendation is to immediately make sure you are in control of both IPv4 *and* IPv6 traffic.

3.5 A Sample Network

When we talk about network we also often see network drawings. The figure 3.3 show a small network with the usual parts. This could be your home network or a small office network.

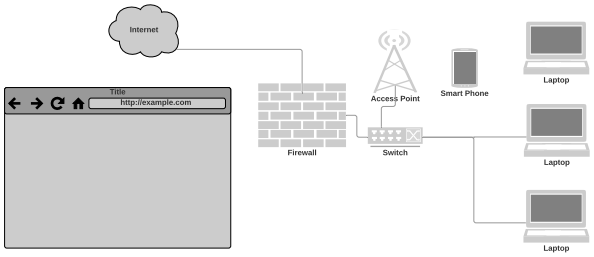


Figure 3.3: Internet browsing

We will not go into each part of this drawing or the network devices found, but note that even though we consider it seperate functions, more features can be built into a small home router, like having the Wi-Fi access point built-in.

Chapter 4

What are Firewalls

Firewalls are network security devices which can inspect traffic and allow or disallow flows to pass through them. They perform this according to a security policy, often called firewall filters or rulesets. Typically they will allow traffic to pass according to basic features such as the direction of traffic flows, the destination or source addresses of the communicating devices.

Firewalls have existed in computing since the 1980s as basic packet filtering firewalls. Originally as software toolkits that were added to Unix systems performing routing functions in networks. Today these have evolved into dedicated hardware devices with electronics and hardware to support high performance routing, while filtering network traffic at *wire speed*.

An enterprise often have multiple computer networks that usually grow over time. These networks often become more complex and have mutually dependent relationship between the many devices. Further these devices may be administered by different groups and have different management strategies.

The following will take a look at a few definitions of the concepts firewalls, before we dive into the features available.

Multiple definitions for firewalls exist, below are a few examples to illustrate some of the variations that exist.

The first book about firewalls used this definition:

> We define a firewall as a collection of components placed between two networks that collectively have the following properties:
>
> - All traffic from inside to outside, and vice-versa, must pass through the firewall.
> - Only authorized traffic, as defined by the local security policy, will be allowed to pass.
> - The firewall itself is immune to penetration.
>
> We should note that these are design goals; a failure in one aspect does not mean that the collection is not a firewall, simply that it is not a very good one.
>
> Source: *Firewalls and internet security; repelling the wily hacker.* by Cheswick and Bellovin (1994)

We will consider this a firewall, but we know today that both inside and outside are meaningless, since we have multiple networks inside, we have partner network connections etc.

Another short definition that encapsulates this is found on Wikipedia, and may suffice in many situations. Again there will typically be multiple networks, zones or areas of the networks with varying degrees of trust.

> In computing, a firewall is a network security system that monitors and controls incoming and outgoing network traffic based on predetermined security rules.[1] A firewall typically establishes a barrier between a trusted network and an untrusted network, such as the Internet.[2]

Source: Wikipedia: https://en.wikipedia.org/wiki/Firewall_(computing)

A very short, but also useful definition is:

> **Firewall Technology:** Mechanism to help enforce access policies about communication traffic entering or leaving networks.

Source: *A reference model for firewall technology and its implications for connection signaling* by Lyles and Schuba (1996)}

I am also fond of this longer and technical definition from RFC4949:

> **$ firewall**
>
> 1. (I) **An internetwork gateway that restricts data communication traffic to and from one of the connected networks** (the one said to be "inside" the firewall) and thus protects that network's system resources against threats from the other network (the one that is said to be "outside" the firewall). (See: guard, security gateway.)
>
> 2. (O) **A device or system that controls the flow of traffic between networks using differing security postures.** Wack, J. et al (NIST), "Guidelines on Firewalls and Firewall Policy", Special Publication 800-41, January 2002.
>
> Tutorial: A firewall typically protects a smaller, secure network (such as a corporate LAN, or even just one host) from a larger network (such as the Internet). The firewall is installed at the point where the networks connect, and the firewall applies policy rules to control traffic that flows in and out of the protected network.
>
> **A firewall is not always a single computer.** For example, a firewall may consist of a pair of filtering routers and one or more proxy servers running on one or more bastion hosts, all connected to a small, dedicated LAN (see: buffer zone) between the two routers. The external router blocks attacks that use IP to break security (IP address spoofing, source routing, packet fragments), while proxy servers block attacks that would exploit a vulnerability in a higher-layer protocol or service. The internal router blocks traffic from leaving the protected network except through the proxy servers. The difficult part is defining criteria by which packets are denied passage through the firewall, because a firewall not only needs to keep unauthorized traffic (i.e., intruders) out, but usually also needs to let authorized traffic pass both in and out.
>
> Source: *Internet security glossary, version 2* Shirey (2007)

In this definition "I" identifies a RECOMMENDED Internet definition while "O" identifies commentary or additional usage guidance.

Markings in bold are mine, and hopefully highlight important parts of this concept. Essentially the devices performs this function by filtering the network traffic, removing unwanted traffic and allowing traffic according to some policy, rule set and configuration.

4.1 Firewalls are part of an Infrastructure Architecture

A very important note is that a firewall is often a collection of components working together. Sometimes there are multiple devices in parallel providing high availability, a cluster of firewalls, and sometimes there are multiple layers of firewalls and filtering devices working together.

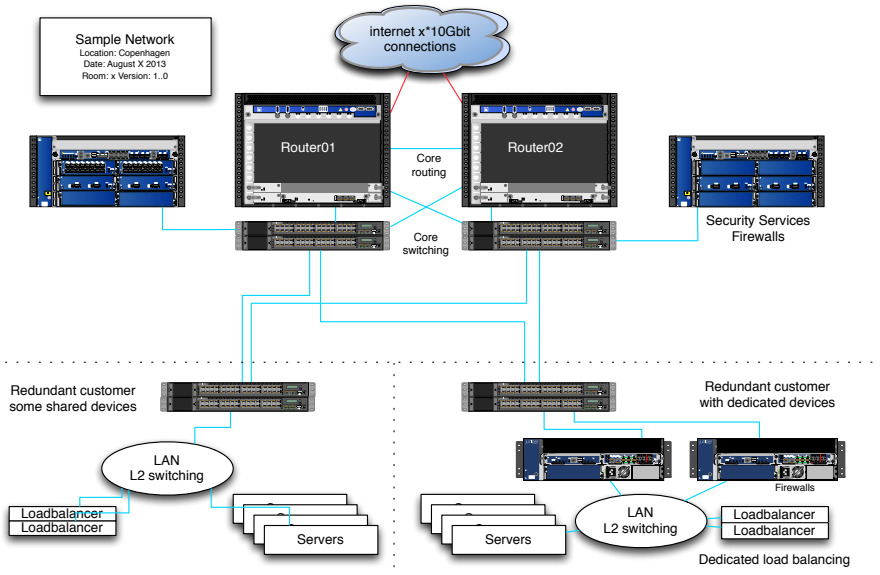


Figure 4.1: The network firewall infrastructure

The figure 4.1 show a number of network devices which can be considered part of the network firewall structure. Packets entering the network from the internet may pass a number of devices which each have their speciality in filtering and firewalling. Further devices working at different layers also have security mechanisms built-in which can help implement the security policy. Email gateways and web proxies are examples we will not cover in this book.

If you can use the features in each layer your firewall can work much more efficiently.

Firewalls are by design a choke point, a natural place to do network security monitoring! Note: Network Security Monitoring is a subject in itself. We recommend the book *Applied network security monitoring* by Sanders and Smith (2014).

Further history about firewalls can be found in articles and documents similar to these early references *The design of a secure internet gateway* Cheswick (1990) and *Firewalls and internet security; repelling the wily hacker. Second edition* Cheswick, Bellovin, and Rubin (2003).

4.2 Network Traffic Filtering History

Throughout the history of the internet we have seen attacks being carried out over the internet, since before 1990 and during the 1990s we got a technology called the firewall. The concept was described in more detail and both toolkits and vendors provided features for common operating systems at the time. The history of firewalls can be researched in places like the first book about firewalls from 1994, which was last updated to second edition in 2003, *Firewalls and internet security; repelling the wily hacker. Second edition.* There is also a companion web site http://www.wilyhacker.com/

One of the early implementers of firewalls Marcus J. Ranum summarized in 2005 *The Six Dumbest Ideas in Computer Security* which includes the always appropriate discussion about default permit versus default deny.

> **#1) Default Permit**
>
> This dumb idea crops up in a lot of different forms; it's incredibly persistent and difficult to eradicate. Why? Because it's so attractive. Systems based on "Default Permit" are the computer security equivalent of empty calories: tasty, yet fattening.
>
> The most recognizable form in which the "Default Permit" dumb idea manifests itself is in **firewall rules**. Back in the very early days of computer security, network managers would set up an internet connection and decide to secure it by turning off incoming telnet, incoming rlogin, and incoming FTP. Everything else was allowed through, hence the name "Default Permit." This put the security practitioner in an **endless arms-race with the hackers**.
>
> Suppose a new vulnerability is found in a service that is not blocked - now the administrators need to decide whether to deny it or not, hopefully, before they got hacked. A lot of organizations adopted "Default Permit" in the early 1990's and convinced themselves it was OK because "hackers will never bother to come after us." The 1990's, with the advent of worms, should have killed off "Default Permit" forever but it didn't. In fact, most networks today are still built around the notion of an open core with no segmentation. That's "Default Permit."
>
> …
>
> The opposite of **"Default Permit"** is **"Default Deny"** and it is a **really good idea**. It takes dedication, thought, and understanding to implement a "Default Deny" policy, which is why it is so seldom done. It's not that much harder to do than "Default Permit" but you'll sleep much better at night.
>
> Source: https://www.ranum.com/security/computer_security/editorials/dumb/

(Bold by me)

I will describe in detail later what we mean when using the term *default deny* policy.

I highly recommend analyzing traffic patterns and let that guide your firewall configuration. Often you will find that a default permit is not needed, and can be replaced by a short list of the systems that should be allowed to communicate.

4.3 The Firewall is Not a Single Device

Usually when people say *firewall* they are not being very specific, they may ask have you opened the firewall, have you restarted the firewall. This can be difficult to work with, especially when you are new to the area.

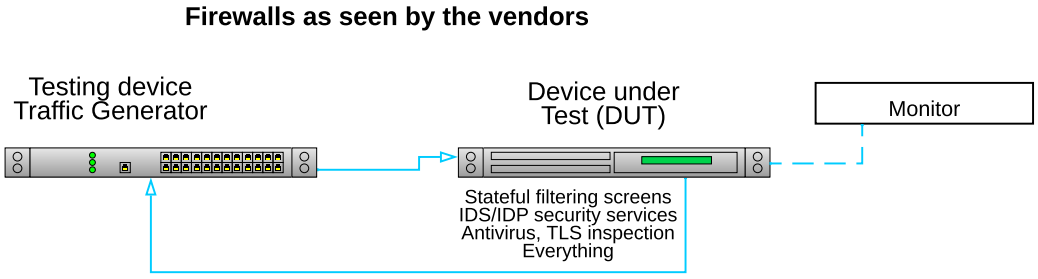


Figure 4.2: Firewalls are often sold as *converged* and one-box solutions

To let me stress an important point: **the firewall devices are not alone**

Security of the network already relies on multiple other devices and functions:

- The firewall devices, your Checkpoint, OpenBSD PF, Juniper, Cisco ASA, Palo Alto etc.
- The routers in front, first line of defense, make use of them
- The switches and routers on the inside, especially VLANs and routing between sites are important
- Also wifi, port security on switches, DHCP snooping, IPv6 First Hop Security, many related features work together

Most server operating systems also include firewall software, so you can have a multi-layered approach to filtering traffic – which is also what we recommend.

4.4 Network Architecture Components

Network paths between systems depend on the use of the devices. We also see that networks often span multiple floors, buildings or even across countries. These networks are often described as Local Area Network (LAN) and Wide Area Network (WAN). In these networks most connections are using Ethernet technologies for connecting across longer distances, and often a LAN today would use Wi-Fi technologies for clients.

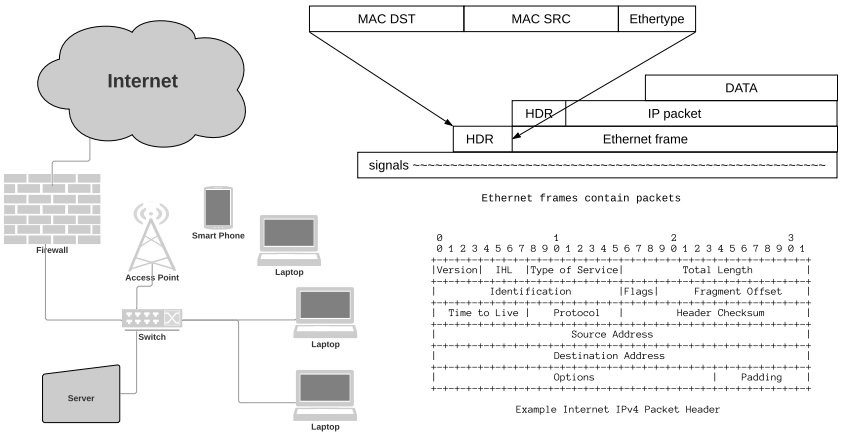


Figure 4.3: Firewalls are often sold as converged and one-box solutions

Shown on the figure 4.3 are a few clients *connected to the internet*. Whenever we provide network services and access networks, our traffic travels over multiple devices. Some of these are layer 2 devices, like access points and switches, while others perform routing.

Expanding this we can see that buildings and offices connected over WAN are a distributed network. This network can quickly grow to thousands of devices and to secure such a complex structure we need to implement filtering.

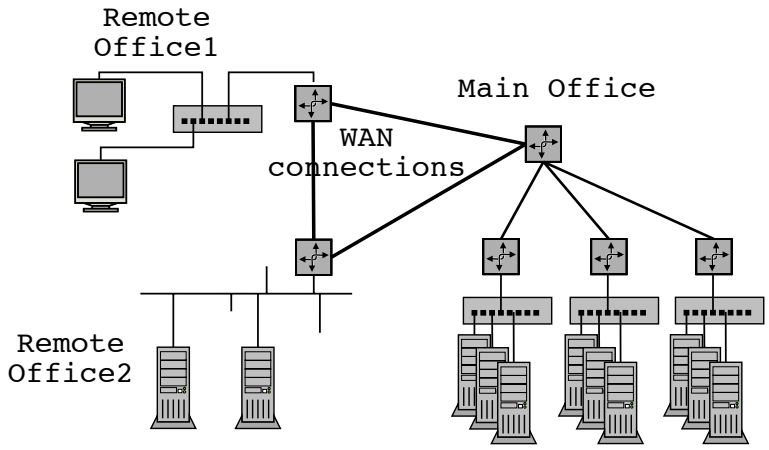


Figure 4.4: WAN Network

If we look further into parts of this structure, for instance a server farm or hosted services we can also trace multiple devices on the network path from the internet to the individual servers. Each of these devices have a feature set and can provide routing, filtering and load balancing to name a few.

Even though they have overlapping feature sets they are often more efficient for specific use-cases. A high end router may transport millions of packets at wire speed, but if the traffic contains attacks it may not be acceptable for the firewall device providing security services to receive it all. Depending on the firewall ruleset it may discard most of the traffic, but will spend resources inspecting it. To reduce the load on the *stateful filtering* devices we propose that routers in front do some basic *stateless filtering*.

The drawing below illustrate some of the features that can be used on each part to reduce the load on the devices used for protecting the network.

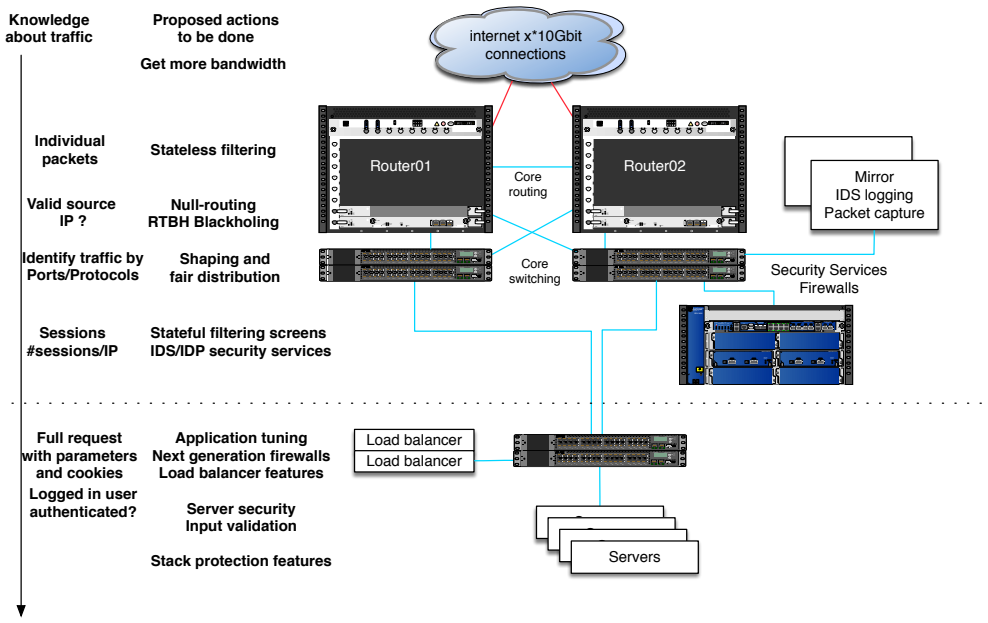


Figure 4.5: The firewall consist of multiple devices providing protection at different layers

The *firewall infrastructure* have lots of features of which a large part is not provided by single devices. The switches that connect systems have *port security*. The Wi-Fi Access Point (AP) devices may provide *network isolation*, rogue access point detection, and other Wi-Fi related security features. The Wi-Fi APs and switches

can provide network segmentation by adopting VLAN assignment and IEEE 802.1x user authentication – dependent on a user directory like LDAP or Active Directory.

We can achieve better network security if we take a more holistic approach to firewalls and what is part of the firewall infrastructure.

Often you can tune and use features like stateless router filters to free up resources in other security devices, which can then handle more traffic and larger attacks than before. Likewise a web load balancer might be better at handling filtering of web requests – since it already might be the endpoint for Transport Layer Security (TLS) encryption, and has the clear text of the request.

Part II: Implementing Firewalls

Chapter 5

Design and Planning of Firewalls

5.1 Design Your Solution

> "A goal without a plan is just a wish." Antoine de Saint-Exupéry

When was the last time you took 3 hours with a whiteboard and designed your network?

To have a secure network does not happen by itself. You need to design your network, while making sure you are within budgets and physical constraints.

There are many resources available about network design, but often you need to decide at least the following:

- Do we need a strictly controlled network, in which there will be more administration
- Do we only have clients which use resources on the Internet or cloud systems
- Do we have a few servers which can share a single Demilitarized Zone aka a segment of the network for servers
- Do we have Wi-Fi – and do we want a guest network with only access to the Internet

When you start asking such questions you will have a better idea of what is needed, and can be done.

Even if you have an existing network, and making changes can be hard – or you do not have resources available to change it. We suggest that you design you dream network, and when you have a chance to make the existing one better.

One of the reasons we enjoy using open source firewall software is that we can design the network according to technical reasons, and not monetary. Since most organisations will have clients, servers and a need for VPN we often suggest having:

- Dedicated client firewall systems
- Dedicated servers firewall systems
- Dedicated VPN servers

One reason this is important is that firewall software *also* has security issues and vulnerabilities. If there is a bug affecting the VPN part of the software, it is much easier to have a service window for the dedicated VPN devices, and not need servers or clients to be offline during the upgrade.

Another main point is that while public IPv4 addresses are limited, and your organisation might only have few – you really should look into implementing IPv6. This will allow you to rethink your network, and perhaps make it more efficient, a cleaner design, more understandable and then it will be easier to secure and harden!

5.2 Address Planning

One part of network security which is often overlook is the address plan for the network. We see many networks that use the private RFC1918 space like 10.0.0.0/8 and then start randomly assigning network prefixes and IP addresses to subnets and devices. This can be harmful and will make future configuration harder.

Before we get into this, lets just review an important concept of Classless Inter-domain Routing (CIDR):

Basic Concept and Prefix Notation

> 3.1 In the simplest sense, the change from Class A/B/C network numbers to classless prefixes is to make explicit which bits in a 32-bit IPv4 address are interpreted as the network number (or prefix) associated with a site and which are the used to number individual end systems within the site. In CIDR notation, a prefix is shown as a 4-octet quantity, just like a traditional IPv4 address or network number, followed by the "/" (slash) character, followed by a decimal value between 0 and 32 that describes the number of significant bits.

Source: RFC4632[1]

So an example might be:

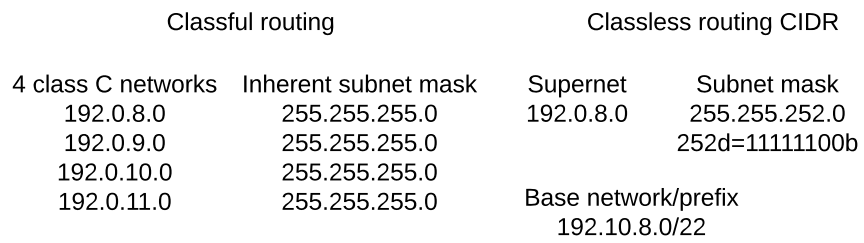


Figure 5.1: Classless Inter-Domain Routing notation

See more about CIDR on Wikipedia https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing.

IPv6 addresses always use CIDR and are written as hexedecimal numbers.

We recommend using the advise from RIPE NCC as a starting point: https://www.ripe.net/publications/ipv6-info-centre/deployment-planning/create-an-addressing-plan/

Which have other resources listed as *Further Reading*:

- Preparing an IPv6 Addressing Plan Manual (SURFnet) https://www.ripe.net/documents/1453/BasicIPv6-Appendix-AddressingPlanHowTo.pdf
- IPv6 Address Planning (Internet Society) http://www.internetsociety.org/deploy360/resources/ipv6-address-planning/
- IPv6 Addressing Overview (Juniper Networks) http://www.juniper.net/documentation/en_US/junos13.2/topics/concept/routing-protocols-ipv6-overview.html#id-10180756
- IPv6 Subnetting – Overview and Case Study (CISCO Support Community) https://supportforums.cisco.com/document/66991/ipv6-subnetting-overview-and-case-study
- IPv6 Subnetting Made Easy (Techxcellence) http://techxcellence.net/2011/05/09/v6-subnetting-made-easy/
- Best Current Operational Practices – IPv6 Subnetting http://bcop.nanog.org/images/6/62/BCOP-IPv6_Subnetting.pdf
- Case Study: An ISP IPv6 Addressing Plan (OTE) https://ripe67.ripe.net/presentations/222-ripe67-yanodd-ipv6-addressing.pdf

5.3 Special Addresses

There are some special addresses in IPv4 and IPv6 which can be used by anyone for private networks.

They are:

- 10.0.0.0 - 10.255.255.255 (10/8 prefix)
- 172.16.0.0 - 172.31.255.255 (172.16/12 prefix)
- 192.168.0.0 - 192.168.255.255 (192.168/16 prefix)

These are originally documented in RFC-1918 *Address allocation for private internets* Rekhter et al. (1996).

NB: not to be used on the Internet

Other special addresses include the documentation prefix, which is intended for examples and documentation:

[1] https://datatracker.ietf.org/doc/html/rfc4632

```
The blocks 192.0.2.0/24 (TEST-NET-1), 198.51.100.0/24 (TEST-NET-2),
and 203.0.113.0/24 (TEST-NET-3) are provided for use in
documentation.
```

Source: RFC5737

```
The IPv6 prefix for documentation purposes is 2001:DB8::/32
```

Source: RFC3849

And you may have noticed that a client computer sometimes use this network:

```
169.254.0.0/16 has been ear-marked as the IP range to use for end node
auto-configuration when a DHCP server may not be found
```

Typically it happens when there is no DHCP server, or it is not functioning.

Chapter 6

Creating Firewall Policies

Firewall policies a description what is allowed and what is not allowed. These policies can be thought of as a list of rules – rule-set but is often augmented by other settings.

The main goal is to allow *good* traffic and block *bad* traffic, but what is bad is up to the organisation. Oftentimes we will not have a textual description of the firewall policies, even though we should, but only have a running configuration based on the syntax of the chosen technology. The policy should be updated using procedures and approval flows, where some request for firewall change is approved and only then implemented.

6.1 Example firewall policy editor OPNsense

If you prefer graphical user interfaces, there are multiple firewall implementations that include sophisticated interfaces, like OPNsense[1]. The screenshot in figure 6.1 is from their excellent documentation.

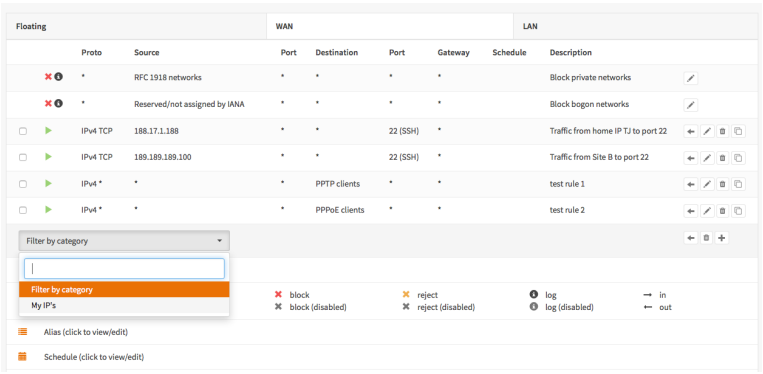


Figure 6.1: Source: OPNsense firewall documentation https://docs.opnsense.org/

Note: OPNsense firewall is based on FreeBSD[2] and FreeBSD PF.

When using graphical interfaces you can easily make changes, but sometimes it can be hard to get an overview of the policy or ruleset as we often call them.

6.2 OpenBSD Example Firewall Configuration

Lets see a firewall configuration, and start with a simple example. We will use the syntax from the OpenBSD[3] Packet Filter (PF) firewall. This firewall technology originates in the OpenBSD operating system, but has later been ported into FreeBSD and Mac OS X.

This language and syntax is in my opinion quite easy to read, even for newcomers.

This configuration is found in `/etc/examples/pf.conf` when installing OpenBSD. Comments were removed, to show only the minimum configuration needed.

[1] https://opnsense.org/get-started/
[2] https://www.freebsd.org/
[3] https://www.openbsd.org/

Minimal OpenBSD PF configuration

```
# Minimal PF firewall policy pf.conf
# Could be used for a server
# This would probably need additional rules to allow administration

set skip on lo

block           # default deny-all rule to block all traffic
                # default behaviour is to drop packets silently
pass out        # allow outgoing traffic establish keep-state

# Default port for SSH is 22/tcp - replace if changing this
# Additionally recommend only allowing SSH administration from specific networks
pass in proto tcp from any to any port ssh
```

The first line tells the firewall to ignore the loopback interface with traffic internally on the system.

The next line is a default block, deny-all rule. This will mark any packets incoming for being thrown away – if not matching any pass rules it will be discarded silently.

The line with pass out allows traffic outgoing from the system. A user or program on the system would be able to communicate to other systems, and the stateful filtering would save information about ongoing sessions.

6.3 Updating Firewall Policies and Configuration

Start by doing this:

- Always keep backup configurations, keep them for a long time, version controlled! Detailed information kept, even after deleting a rule. Tool recommendation: Oxidized[4] or use the built-in features for copying the configuration in your product
- Even better - automate your firewall and filtering configuration changes roll out a change, and easy to revert changes. Tool recommendation for filtering routers: Ansible[5]
- Always document all parts, especially VPN connections - partner company and contact information

Then

- Review your configuration regularly, this may be hard as the network becomes large and complex
- Firewall and filtering rules are more than just the technical parts - also authorized by, date inserted/removed

[4] Rules_Full-opensense.png
[5] https://en.wikipedia.org/wiki/Ansible_(software)

Chapter 7

Core Firewall Features

7.1 Network Traffic are Packets

When we use the Internet we are sending a lot of *packets* .

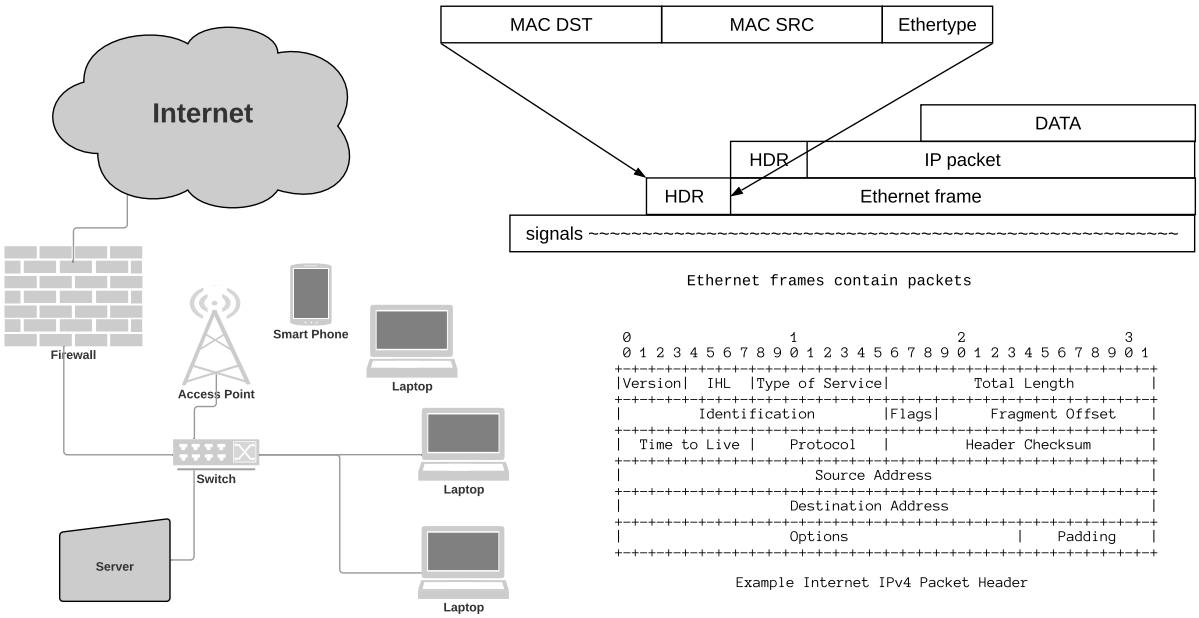


Figure 7.1: Data on the Internet are sent as data packets

7.2 What Features Constitute a Firewall

Today we use the term firewall in a broad sense, describing devices with many features. These devices are often being branded and sold as firewall products. This book will only consider a narrow subset of these feature, but later books may expand on the subject.

We will consider a firewall device a system containing network hardware, control systems and software. These devices are typically connecting networks with different security requirements, or owned and operated by different entities.

Such a device is an excellent place to enforce security policies. The following core features are part of filtering traffic in TCP/IP networks and is considered a firewall:

- Filtering incoming traffic - requests coming in from a network via a network interface is evaluated with a security policy
- The security policy may decide to deny or allow the network traffic to continue. Often these mean, deny that a response might be generated to the source of the request.

- The device may log requests, both those allowed, but also the denied requests

7.3 Discard, drop or reject

An important part of the security policy is deciding if we silently drop or discard the packets that are denied entry, or if we send back a rejection in form of a message saying that the traffic is denied. There are multiple reason why we would like to send back information. Often this will quickly allow the sending system to know, traffic was denied. This will enable quick response to a user - access was denied to some site, and we avoid long time outs. On the other hand an attacker could also use this information to scan our IP ranges and map our networks. For this reason we will often allow inside networks to get a response, using a reject policy, while internet traffic we deny will be silently discarded – but may still be logged.

7.4 Types of Firewall Filtering

The functions in firewall used to filter traffic can roughly be categorized into these functions:

- Packet filters - stateless filtering where a single packet is evaluated; discarded or allowed
- Stateful filtering/firewalling - where devices can hold state information about sessions or connections, initial packet received, established TCP connections etc.
- Application level features, such as filtering specific protocols DNS, HTTP etc. These can be small modules for simple protocols, or complex features

This book will focus on the lower levels of the IP stack which are the first two bullets in the list above.

The OSI model for networks is a common tool for teaching network communication and describe how layering can divide a complex problem into parts, that can be solved independently. While our network hardware is not implemented strictly after this model, we still use the descriptions. Below is the OSI model shown in figure 7.2, along with the layers in the TCP/IP. There is not a one to one mapping, but the design of TCP/IP can also be considered layered and the layers match approximately.

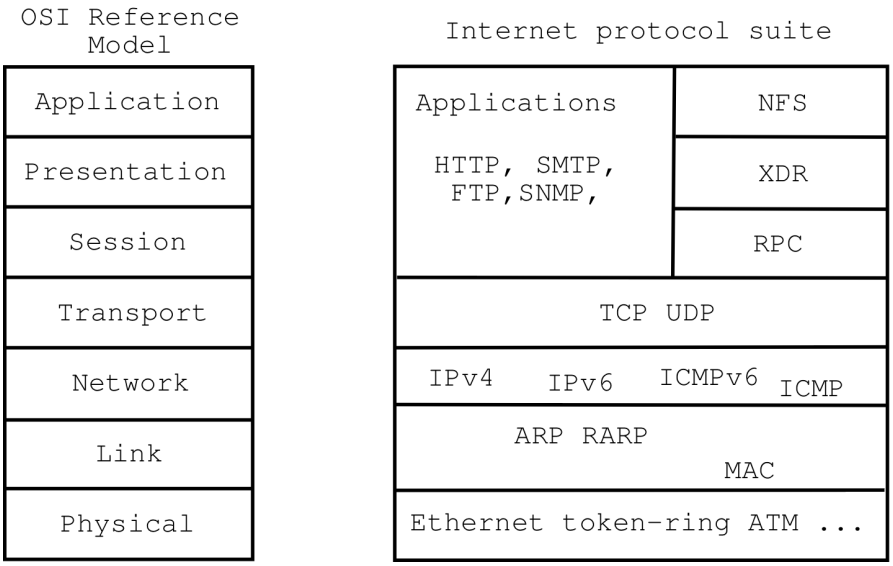


Figure 7.2: OSI model compared to the IP suite

Usually a firewall would not consider the hardware near layers. Filtering IPv4 typically start when the Address Resolution Protocol (ARP) has discovered the MAC hardware address for the technology used. So only layer 3 and upwards - the IP layer is considered.

Lower levels can be configured in other systems like switches and Wi-Fi controllers where Virtual LAN (VLAN) and *client isolation* features control lower layers of communication.

7.5 Firewalls and IPv4/IPv6

This document will include examples with both IPv4 and IPv6 protocols. The main difference between these are that IPv6 resolves hardware addresses using the Neighbor Discovery Protocol (NDP) which is part of

ICMPv6.

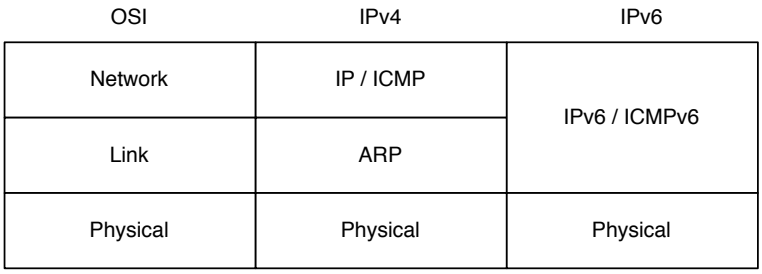


Figure 7.3: IPv4 uses ARP, while IPv6 uses NDP

Some firewall devices, like the OpenBSD PF technology used in this book require us to specifically allow the ICMPv6 packets for IPv6 traffic to work. We will describe this problem in more detail while creating a simple firewall policy for a single system in chapter 9.

7.6 Basic Packets

When transmitting data across the internet with TCP/IP we send packets at the lowest levels, and below that some networking technology like Ethernet frames.

These packets contain what is called a header, like a postcard. The header for Internet Protocol version 4 is shown here:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IPv4 packet header Source: RFC0791

The most common fields used when filtering are:

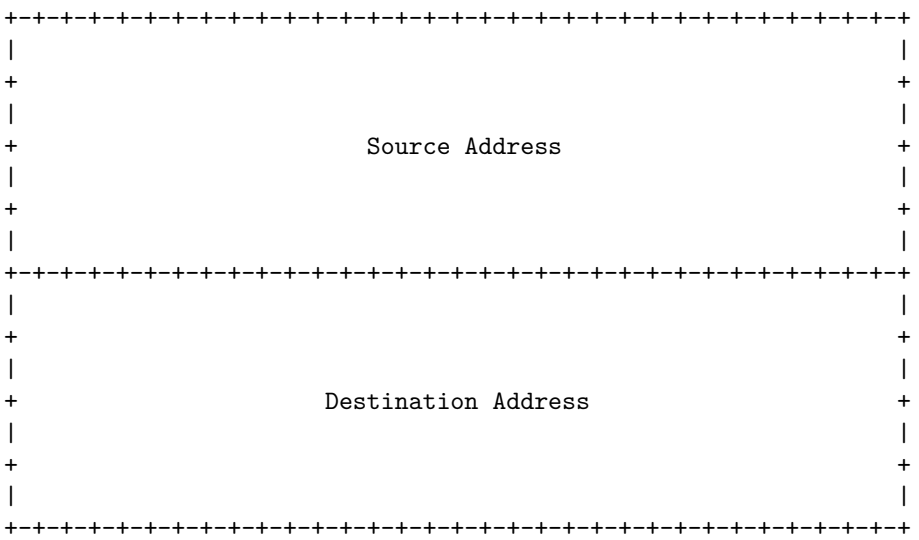
- protocol – do we allow ICMP, TCP, UDP into this part of the network
- Source address – do we want to allow traffic from this IP, which might also be spoofed
- Destination address – which system is this for, a public server with HTTP or a private service

The direction of traffic is also important, is this traffic going into our network or going out. For example a web server with HTTP service will receive a lot of requests from the internet, any source address allowed and destined for the IP address of the server with port 80/tcp. This same server will typically NOT initiate connections to the Internet at large, so it will be quite safe to block most outgoing traffic from this server.

Direction of traffic is usually configure per rule and using the network interfaces on the firewall device. So traffic will often traverse an inside interface and an outside interface.

The IPv6 header is more simple, has a fixed structure – but addresses are 128-bit instead of 32-bit:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version| Traffic Class |           Flow Label                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Payload Length        |  Next Header  |   Hop Limit   |
```



IPv6 packet header Source: RFC2460

7.7 TCP, UDP and Application headers – Flows

After these headers applications will most likely use either UDP or TCP, and they have headers which include port numbers. A port number is often from a list of well-known services, as defined and registered by Internet Assigned Numbers Authority (IANA).

These registration include services such as:

- 53/udp and 53/tcp Domain Name System (DNS)[1] – name resolution
- 80/tcp Hypertext Transfer Protocol (HTTP)[2]
- 443/tcp Hypertext Transfer Protocol Secure (HTTPS)[3]

See the full list of registrations at IANA:

https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

Protocols that use TCP is a special case, since the packets are part of a *connection* or *TCP flow*. This traffic will be initiated from a source using a *TCP handshake*. We will cover this in detail in chapter 8.

Note: when a TCP connection is made the connection is identified by a 5-tuple, consisting of these 5 parameters:

- Source IP Address: The IP address of the device initiating the connection.
- Source Port: The port number on the initiating device.
- Destination IP Address: The IP address of the device receiving the connection.
- Destination Port: The port number on the receiving device.
- Protocol: In this case, TCP (Transmission Control Protocol).

So when your browser makes a connection with HTTP using TCP, it uses a high numbered source port and initiates a connection towards the destination port 80 or 443 if using HTTPS. This connection will receive return packets from source port 80 towards your high numbered port.

7.8 Example Firewall Devices

In this resource I will refer to examples of firewall devices, which can be both firewalls but also switches, routers and other network devices. I will try to refrain from using the broad term firewalls, since these devices all work together to provide network security.

Some technologies I will use for examples are:

- Debian Uncomplicated Firewall (UFW) an easy to configure firewall software for servers
- OpenBSD which is an open source operating system focused on security with a built-in firewall software and implementing various core network protocols like BGP and OSPF

[1] https://en.wikipedia.org/wiki/DNS
[2] https://en.wikipedia.org/wiki/HTTP
[3] https://en.wikipedia.org/wiki/HTTPS

- Arista EOS providing switching and routing, as well as filtering
- Cisco firewalls and routers, being some of the most popular firewalls for many years, and their IOS and now IOS-XR are running large parts of the internet
- Juniper Junos which also runs a large part of the internet, and they provide devices which are routers and firewall devices

Examples of stateless Packet filters are:

Vendor & Variant	Terminology
Arista EOS	Access Control List (ACL) IPv4 and IPv6
Cisco IOS	Access Control List (ACL) IPv4 and IPv6
Juniper Junos	Stateless Firewall Filters

Note the name of the stateless filter on Junos.

Stateful firewalling examples are:

Vendor & Variant	Terminology
OpenBSD PF	default pf(4) filters packets statefully
Cisco ASA	default to stateful filtering
Juniper SRX	Security Policies, but can use stateless filters at the same time

7.9 Next-gen Firewalls

I will refrain from using the term next-gen / next-generation which many firewall and network vendors use. This term has very little meaning, since we have moved far beyond having generations of firewall technologies and this distinction, if ever relevant, is not anymore.

You should consider your use-cases more when buying devices, and while one firewall might be perfect for protecting users and office environments from threats the same devices may be of little use in a hosting and data center context.

Further many firewal vendors include a multitude of related application level features. These may include: Application intelligence, Deep packet inspection (DPI), Proxy – like DNS proxy, certificate inspection

Different vendor terms also exist for these feature, but in general the main feature is being able to decode packets which implement application protocols at a higher level.

7.10 Firewalls NAT and High Availability

Network Address Translation (NAT)[4] is a feature available in most firewall products. Unfortunately this feature is also configured very differently in those products.

Similarly features like failover, high availability are often product specific, and we recommend consulting the vendor documentation for those.

For this reason, and to keep the book simple, we will mostly ignore NAT – except in examples showing complete firewalls.

[4] https://en.wikipedia.org/wiki/Network_Address_Translation

Chapter 8

Stateless vs Stateful Filtering

8.1 Filtering ICMP

A very common program to use in networking is the Ping program[1].

Using this tool you can send a ping packet, using the Internet Control Message Protocol(ICMP) with a special type 8 and code 0. If the network is connected all the way to the destination, and it is allowed – the system would send a new return packet with ICMP type 0 and code 0.

These ICMP Echo request and reply messages would all have a similar structure as documented:

```
Echo or Echo Reply Message

0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Identifier          |        Sequence Number        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Data ...
+-+-+-+-+-
```

Source: RFC 792 September 1981

The value of the data field is returned in the response, and can be used for checking communication links for suspected problems with specific values. Different operating system implementations of the ping program has selected different values for the data, both length and content.

Another typical use of ping is the check the Path MTU (MTU) – Maximum Transmission Unit, the largest packets that can be sent.

If we want ICMP Echo and Reply to work in our firewall, we could add two rules:

- Allow outgoing to any icmp echo request to any destination
- Allow incoming to any icmp echo reply to any destination

This kind of filtering is called stateless filtering , see RFC2401 by Kent and Atkinson (1998) and others. This can be done with routers, switches and dedicated *firewall devices*.

Note: there are multiple ICMP messages that should *NOT* be filtered, as this would make error reporting harder, and clients will experience longer time outs while waiting for responses. Our recommendation for allowing ICMP messages includes:

- 3 unreach Destination unreachable
- 4 squench Packet loss, slow down
- 11 timex Time exceeded – often if there is a routing loop, but also traceroute[2]

[1] https://en.wikipedia.org/wiki/Ping_(networking_utility
[2] https://en.wikipedia.org/wiki/Traceroute

- 12 paramprob Invalid IP header

For IPv6 we recommend taking special care and allow at least some of these:

- 1 unreach Destination unreachable
- 2 toobig Packet too big
- 3 timex Time exceeded
- 4 paramprob Invalid IPv6 header

and critically you *NEED* to allow – otherwise no IPv6 traffic will be sent:

- 135 neighbrsol Neighbor solicitation
- 136 neighbradv Neighbor advertisement

This is used by Neighbor Discovery Protocol (NDP) that replaces IPv4 Address Resolution Protocol (ARP)

See section 11.9 for more information about this change from IPv4.

8.2 Example Stateless Filtering using Cisco IOS/Arista Syntax

While routers and switches don't typically allow for full stateful filtering, they can match packets expected to belong in an established session. Packets without the TCP SYN flag, but with the flags ACK (and RST) are acknowledgement and reset packets.

Both network vendors Arista EOS and Cisco IOS, among others, allow us to specify this behaviour with a keyword *established*. This could be useful when having servers that are connected to the internet, but are not expected to initiate connections themselves directly to the internet.

So it would be easy to allow return

```
access-list 100 permit tcp any any established
```

When this is specified, the system does NOT allow incoming TCP SYN, and thus we can assure that no TCP sessions are created incoming. If we were to allow from a specific network we could augment this with:

```
access-list 100 permit tcp any any established
!--- Permits www traffic from 10.0.42.10 management machine to 192.0.2.5 system
access-list 100 permit tcp host 10.0.42.10 host 192.0.2.5 eq ssh
access-list 100 deny ip any any
```

Note: these examples are not fully developed, and should also be applied on an interface to become active. A more full example could be the Secure BGP Template from Team Cymru [https://team-cymru.com/community-services/templates/secure-bgp-template/]

8.3 Example Stateless filtering using Junos IOS Syntax

Stateless firewall filter throw stuff away

```
hlk@MX> show configuration firewall filter all | no-more
/* This is a sample */
inactive: term edgeblocker {
    from {
        source-address {
            198.51.100.0/24;
            203.0.113.0/24;
        }
        destination-address {
            192.0.2.0/28;
        }
        protocol [ tcp udp icmp ];
    }
    then {
        count edge-block;
        discard;
    }
}
```

This implement what can be termed an edge-blocker – block traffic as soon as possible, on the edge of the network. So it will *Block all traffic from specific source networks* {#edgeblocker} – and will be able to handle *wire speed* traffic, as quickly as the network interface can deliver them, using as few resources as possible.

Hint: can also leave out protocol and then it will match all protocols

This can also be used for an allow list. Say you have a part of the network with dedicated web services using only a few ports and TCP. Then you could program a short list of destination addresses into a few rules:

```
term some-server-allow {
    from {
        destination-address {
            192.0.2.0/24;
        }
        protocol tcp;
        destination-port [ 80 443 ];
    } then accept;
}
term some-server-block-unneeded {
    from {
        destination-address {
            192.0.2.0/24;
        }
        protocol-except icmp;
        }
    then {
        count some-server-block;
        discard;
    }
}
```

This will implement very *Strict filtering* for some web servers, still stateless! {#strict-web-filtering}

This would a great way to ensure no other traffic apart from TCP traffic with these specific destination ports reach into the network. This can be done statelessly and will conserve resources on the next firewall devices. They will have less traffic to inspect.

Further configuration is available on these devices, such as limiting traffic with specific protocols to a percentage of the traffic. Our recommendation is to measure normal traffic levels and then design an input policy for devices which might contain these elements, or more.

Initially all we had was stateless filtering, but today we have something else available – and often in multiple devices on the network path.

8.4 What is a State Anyway

A state when discussing firewalls is generally information about a connection between endpoints. The state allows the firewall to allow responses to traffic sent back in through the firewall. This allows a much more fine grained and efficient firewall.

Without state is is hard to create a secure a rule-set, but with state we can allow traffic out, and the state table will remember which packets to allow as responses.

So in a stateless system we could allow clients to send out a HTTP request over a TCP connection, from a dynamic high numbered source port towards a destination HTTP server on port 80 (or 443 if HTTPS). This would then results in a need for a rule allowing the servers - all the servers on the internet to send return traffic from port 80 (or 443) back.

Essentially this would be two rules, like: * outgoing allow from any source port 32000-65535 to any destination port 80 * incoming allow from any source port 80 to any destination port 32000-65535

The problem would be that any system on the Internet would be able to send from the same source port. It is only by convention that clients send fra a dynamic high port. So the security of this simple rule set would not be very high.

If on the other hand the stateful firewall does something like:

- Policy: outgoing allow from any source port 32000-65535 to any destination port 80
- State table: add entry for client source IP, client source port, destination server IP, destination server port

Then it can match any return traffic, and allow it – while anything else from random source IP addresses are rejected/discarded. This is what a stateful filtering system/firewall does. Notice the events, enforce a more narrow and strict policy.

Unfortunately this also takes up computing power, requires the table to be saved in memory and accessed for each and every packet refceived! This can be expensive in terms of computing power and if there is a high speed attack may exhaust the capabilities of the firewall devices.

A common metric for a firewall is the number of concurrent sessions to handle, typically in the millions for an enterprise firewall.

A small scale exhaustion attack, a single CPU in a sending system can send 14 Million packets per second, so can often fill up the state table with half-open TCP connections by sending SYN packets *initiating a connection*.

The main defense against this is to use SYN cookies[3] or expiring half-open connections more aggressively from the table.

8.5 TCP States

The full state machine for TCP connections can be seen in the figure below:

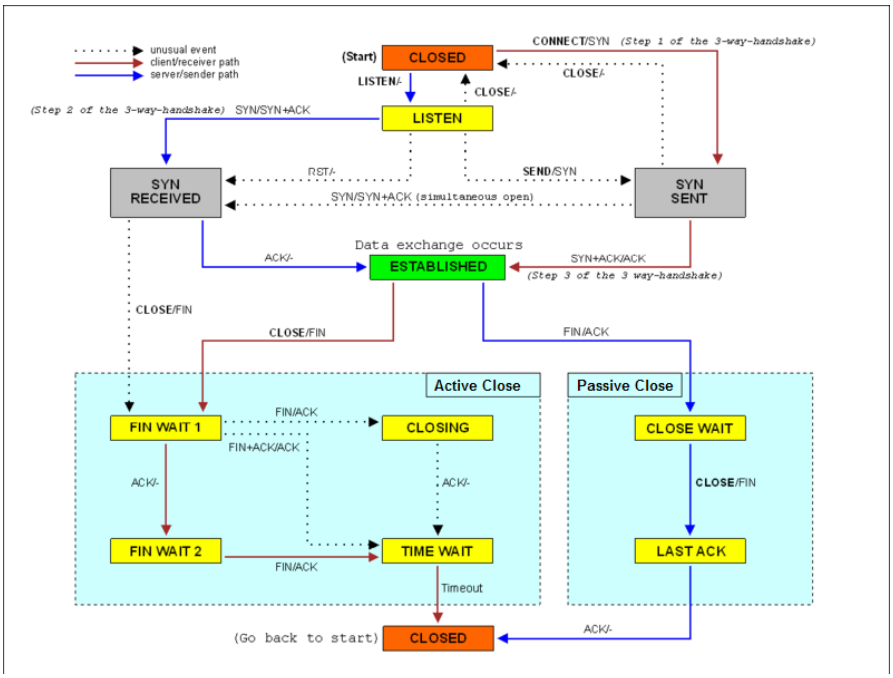


Figure 8.1: The TCP state Diagram

Note: source of this diagram is https://en.wikipedia.org/wiki/File:Tcp_state_diagram.png licensed under the Creative Commons Attribution-ShareAlike 3.0 License. See original for further information.

Since there is extra work involved with stateful filtering, we recommend creating a strategy involving a mix of stateless and stateful filtering.

8.6 Building a Stateless Filtering Strategy

Putting the knowledge into use we can now start to develop a filtering strategy for our network, based on our need.

- The address plan divides our server networks into TCP services, UDP services and one for mixed TCP/UDP

[3] https://en.wikipedia.org/wiki/SYN_cookies

- ICMP traffic is a maximum of 5% of bandwidth. So for a Gigabit connection of 1000Mbps we would allow only 50Mbps to be ICMP
- If UDP is only used sparringly we would allocate a percentage for this. If this would be 10% then we would allow only 100Mbps of UDP traffic
- The services allowed from the internet would have specific ports, and we can add these to stateless filter ACLs as shown above.

What this would ensure is that fewer packets reach the statefull filtering boxes. These would be less likely to be overloaded by DDoS traffic, and would be able to spend more resources doing stateful filtering.

FIXME more input from the recent presentations from hacktivity

8.7 Stateful Filtering with DNS example

Another typical state example is a DNS query, in which a client sends a DNS packet towards a DNS server. Typically this is sent within a UDP packet, but may also be a session of TCP packets.

Example, sending a request from a client to server, and observing the responses:

```
# tcpdump -r dns-example.cap -vvvv
reading from file dns-example.cap, link-type EN10MB (Ethernet), snapshot length 262144
11:36:21.365672 IP (tos 0x0, ttl 64, id 40927, offset 0, flags [none], proto UDP (17), length 61)
    10.137.0.25.58213 > 10.0.42.1.domain: [bad udp cksum 0x3edd -> 0x5f77!] 13195+ A? www.example.com.
11:36:21.679014 IP (tos 0x0, ttl 62, id 11756, offset 0, flags [none], proto UDP (17), length 77)
    10.0.42.1.domain > 10.137.0.25.58213: [udp sum ok] 13195 q: A? www.example.com. 1/0/0 www.example.
    [1d] A 93.184.216.34 (49)
11:36:21.681505 IP (tos 0x0, ttl 64, id 41080, offset 0, flags [none], proto UDP (17), length 61)
    10.137.0.25.40334 > 10.0.42.1.domain: [bad udp cksum 0x3edd -> 0x440d!] 31180+ AAAA? www.example.c
11:36:22.642970 IP (tos 0x0, ttl 62, id 43779, offset 0, flags [none], proto UDP (17), length 89)
    10.0.42.1.domain > 10.137.0.25.40334: [udp sum ok] 31180 q: AAAA? www.example.com. 1/0/0 www.examp
    [1d] AAAA 2606:2800:220:1:248:1893:25c8:1946 (61)
11:36:22.643334 IP (tos 0x0, ttl 64, id 41895, offset 0, flags [none], proto UDP (17), length 61)
    10.137.0.25.59835 > 10.0.42.1.domain: [bad udp cksum 0x3edd -> 0xf859!] 34386+ MX? www.example.com
11:36:22.734874 IP (tos 0x0, ttl 62, id 24717, offset 0, flags [none], proto UDP (17), length 117)
    10.0.42.1.domain > 10.137.0.25.59835: [udp sum ok] 34386 q: MX? www.example.com. 0/1/0 ns: example
    [1h] SOA ns.icann.org. noc.dns.icann.org. 2021090203 7200 3600 1209600 3600 (89)
```

The reason for showing this example is to show some of the details of packet. This dump from a common tool, Tcpdump[4] show a long list of parameters.

This can be very confusing, but show the many opportunities for filtering traffic. It can be done on IP address source and destination, the port numbers – destination port for DNS is typically 53/udp for client traffic – while servers communicate using 53/tcp. The tool tcpdump in the example show *domain* for this port.

The packets are from a source IP: 10.137.0.25 towards a DNS server at IP: 10.0.42.1 port 53 (domain) with UDP packets, and tries to look up the name *www.example.com* using both IPv4 query for A records in DNS, and IPv6 Quad-A (AAAA). The responses can be seen having the source and destination IP/port reverse – answering the query.

A stateful firewall would recognize this outgoing UDP packet and expect a response to be returned, and thus allowed in.

If a device can understand the specific protocol, in this case DNS it can even look into the data – information about names and web site hostname, and filter based on the content. This is often called *deep inspection* and we will not go into more details.

We will though show a sample DNS packet deciphered using Scapy[5] which is a Python language library that allows construction and sending of network traffic:

```
###[ DNS ]###
           id= 13195
           qr= 1
           opcode= QUERY
```

[4] https://en.wikipedia.org/wiki/Tcpdump and https://www.tcpdump.org/
[5] https://scapy.net/

```
aa= 0
tc= 0
rd= 1
ra= 1
z= 0
ad= 0
cd= 0
rcode= ok
qdcount= 1
ancount= 1
nscount= 0
arcount= 0
qd
 |###[ DNS Question Record ]###
 |  qname= 'www.example.com.'
 |  qtype= A
 |  qclass= IN
an
 |###[ DNS Resource Record ]###
 |  rrname= 'www.example.com.'
 |  type= A
 |  rclass= IN
 |  ttl= 86400
 |  rdlen= 4
 |  rdata= '93.184.216.34'
ns= None
ar= None
```

Using tools like these working with network packets become much easier, and afterwards this allow more secure filtering policies to be specified and implemented.

8.8 Recommend Wireshark

If you want to learn more about networks and protocols we recommend working with the Wireshark graphical application[6]. Preferably you would use the documentation from the web site, or maybe a book like the excellent one *Practical packet analysis, 3rd edition*

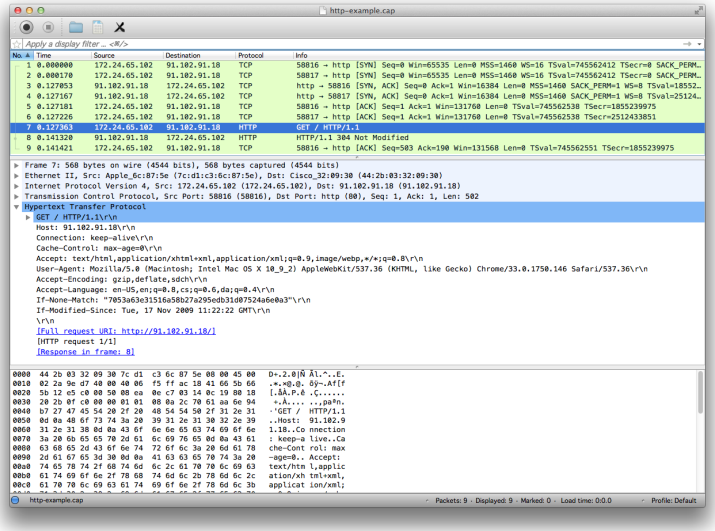


Figure 8.2: Wireshark showing captured HTTP sesssion

[6] https://www.wireshark.org/

Part III: Firewall Examples

Chapter 9

Protect a System: Basic Filtering

Protecting a single device with a firewall is often easy, and should always be done. All devices should protect their kernel, operating system and services. Especially if they contain important data.

When saying easy, it is because:

- A laptop that is used on multiple network should often *not* provide services for *the network*, so should have a very restrictive network policy – nothing is allowed in. Usually client operating systems can use the vendor provided and configured firewall. Examples include Microsoft Windows, Apple Mac OS X
- A server that has some data and a function will provide specific services to other systems. The people responsible for this service should know who the consumers of the service are. So a server can be configured with a default deny policy and opened for the consumers for the specific services used

9.1 Databases

An example might be a database server using MySQL/MariaDB[1]

This database server has an IP address and a port for the MySQL service – by default 3306/tcp.

When another system, the database client want to access the data it will connect to this port, and depending on the security policy it should be allowed or not. So to create a policy for this server and service one would define:

- Allowed source IP addresses, either hosts addresses or network prefixes/subnets. Example 192.0.2.0/24
- Port 3306
- Protocol TCP

Additionally a common port for this server would be the database administrator (DBA) which may login to the system using Secure Shell:

- Allowed source IP addresses, 198.51.100.10 and 198.51.100.11
- Port 22
- Protocol TCP

9.2 Example Debian UFW Firewall Configuration

Most Linux distributions include firewall capabilities with the operating system, but the management of these differ. Since we like the Debian distribution, and Ubuntu being a Debian based operating system we can use the *Uncomplicated Firewall*[2] as an example.

The firewall filtering is enabled by a single command:

```
root@debian01:~# apt install ufw
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

[1] https://en.wikipedia.org/wiki/MySQL
[2] https://en.wikipedia.org/wiki/Uncomplicated_Firewall

```
The following NEW packages will be installed:
ufw
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 164 kB of archives.
After this operation, 848 kB of additional disk space will be used.
Get:1 http://mirrors.dotsrc.org/debian stretch/main amd64 ufw all 0.35-4 [164 kB]
Fetched 164 kB in 2s (60.2 kB/s)
...
root@debian01:~# ufw allow 22/tcp
Rules updated
Rules updated (v6)
root@debian01:~# ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup
root@debian01:~# ufw status numbered
Status: active
To
--
[ 1] 22/tcp
[ 2] 22/tcp (v6)
Action
------
ALLOW IN
ALLOW IN
From
----
Anywhere
Anywhere (v6)
```

Additionally UFW can be controlled using your favourite deployment technology. Example Ansible snippets

```
- name: Enable and deny everything with UFW
  ufw:
    state: enabled
    policy: deny

- name: Set logging
  ufw:
    logging: 'on'

# ufw supports connection rate limiting, which is useful for protecting
# against brute-force login attacks. ufw will deny connections if an IP
# address has attempted to initiate 6 or more connections in the last
# 30 seconds. See  http://www.debian-administration.org/articles/187
# for details. Typical usage is:
- ufw:
    rule: limit
    port: ssh
    proto: tcp

# Allow OpenSSH. (Note that as ufw manages its own state, simply removing
# a rule=allow task can leave those ports exposed. Either use delete=yes
# or a separate state=reset task)
- ufw:
    rule: allow
    name: OpenSSH
    port: 22

- name: Allow all access to tcp port 80 HTTP
  ufw:
    rule: allow
    port: '80'
```

```
    proto: tcp
- name: Allow all access to tcp port 443 HTTPS
  ufw:
    rule: allow
    port: '443'
    proto: tcp

# Note that IPv6 must be enabled in /etc/default/ufw for IPv6 firewalling to work.
- name: Deny all traffic from the IPv6 2001:db8::/32 to tcp port 25 on this host
  ufw:
    rule: deny
    proto: tcp
    src: 2001:db8::/32
    port: '25'
```

Inspired by https://docs.ansible.com/ansible/2.9/modules/ufw_module.html

9.3 Example OpenBSD PF Firewall Configuration

Lets see a firewall configuration, and start with the most simple example. We will use the syntax from the OpenBSD Packet Filter firewall. This firewall technology originates in the OpenBSD operating system, but has later been ported into FreeBSD and Mac OS X.

This language and syntax is in my opinion quite easy to read, even for newcomers.

I have removed comments, to show only the minimum configuration needed. **Minimal OpenBSD PF configuration** {#listing:pf-first}

```
# Minimal PF firewall policy pf.conf
# Could be used for a server
# This would probably need additional rules to allow administration

set skip on lo

block           # default deny-all rule to block all traffic
                # default behaviour is to drop packets silently
pass out        # allow outgoing traffic establish keep-state

# Default port for SSH is 22/tcp - replace if changing this
# Additionally recommend only allowing SSH administration from specific networks
pass in proto tcp from any to any port ssh
```

The first line tells the firewall to ignore the loopback interface with traffic internally on the system.

The next line is a default block, deny-all rule. This will mark any packets incoming for being thrown away – if not matching any pass rules it will be discarded silently.

The line with pass out allows traffic outgoing from the system. A user or program on the system would be able to communicate to other systems, and the stateful filtering would save information about ongoing sessions.

To allow administration the first port to allow in is typically OpenSSH - Secure Shell administration. This can be enabled for all addresses as shown, but it is often better to define a list of prefixes, networks, that are allowed to login through this service.

This configuration can be enabled on OpenBSD using the following commands, as root:

```
# cp /etc/example/pf.conf /etc/pf.conf
# vi /etc/pf.conf     # removing the lines you don't need
# rcctl enable pf
```

- <1> First copy the sample file into the right place
- <2> Optionally edit the file using `vi` or another editor
- <3> Enable PF on next boot of the system

This configuration based on the example found in `/etc/examples/pf.conf` when installing OpenBSD.

The main source of information about this firewall is the OpenBSD PF - User's Guide https://www.openbsd.org/faq/pf/ which also includes this list of commands that are very useful:

```
# pfctl -f  /etc/pf.conf     # Load the pf.conf file
# pfctl -nf /etc/pf.conf     # Parse the file, but don't load it, check syntax
# pfctl -sr                  # Show the current ruleset
# pfctl -ss                  # Show the current state table
# pfctl -si                  # Show filter stats and counters
# pfctl -sa                  # Show everything it can show
```

9.4 Routers and Network Devices

Routers often have a control plane and a data plane. The first is controlling the device, and provides management, while the data plane is forwarding and routing the packets. The data plane is mostly hardware based on high-end devices, and can forward at full wire speed.

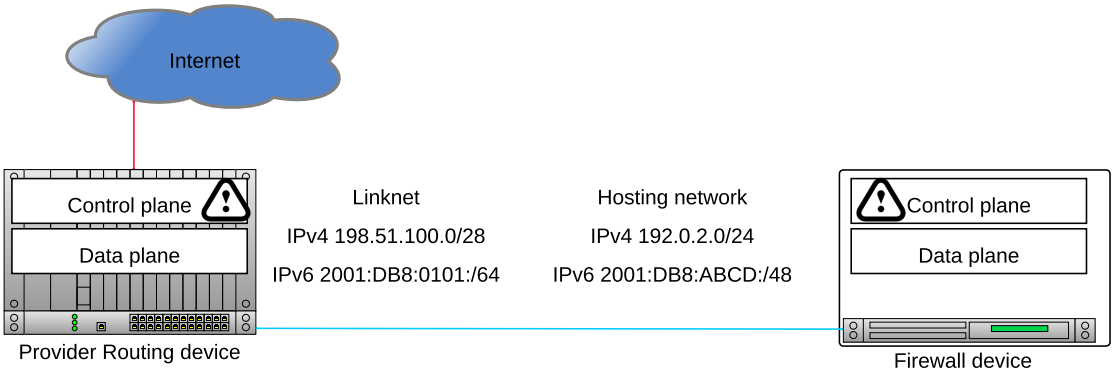


Figure 9.1: Modern network devices have control and data planes

Control-Plane Access Control Lists (CP-ACL) should be installed on network devices, as there have been multiple vulnerabilities in a multitude of products.

An example from a Cisco router using secure shell is shown below with a simple standard Access Control List (ACL) named 22 and then referenced for the virtual terminal (vty) secure shell (ssh):

```
ip access-list standard 22
 10 permit 192.0.2.2
 20 permit 172.13.22.10
 30 permit 192.168.0.10
 40 permit 203.0.113.10
line vty 0 4
 access-class 22 in
 transport input ssh
```

This will reduce the likelihood that an attacker can gain access to the administration using leaked username and passwords or because of vulnerabilities in the software.

Chapter 10

Protect Servers with a DMZ

10.1 Segmented Networks

The previous chapter showed a couple examples of firewall rules, and often these are based on IP prefixes, address ranges. We also discuss a bit about address plans in Chapter 5.

These plans are usually implemented using multiple network segments[1], segmented network – or just network segmentation[2]

Advantages

- Reduced congestion: On a segmented network, there are fewer hosts per subnetwork and the traffic and thus congestion per segment is reduced
- Improved security:
- Broadcasts will be contained to local network. Internal network structure will not be visible from outside.
- There is a reduced attack surface available to pivot in if one of the hosts on the network segment is compromised. Common attack vectors such as LLMNR and NetBIOS poisoning can be partially alleviated by proper network segmentation as they only work on the local network. For this reason it is recommended to segment the various areas of a network by usage. A basic example would be to split up web servers, databases servers and standard user machines each into their own segment.
- By creating network segments containing only the resources specific to the consumers that you authorise access to, you are creating an environment of least privilege[1][2]
- Containing network problems: Limiting the effect of local failures on other parts of network
- Controlling visitor access: Visitor access to the network can be controlled by implementing VLANs to segregate the network

Source: https://en.wikipedia.org/wiki/Network_segmentation

While network segments used to be physical, with seperate cables and switches, today we use *virtual LANs* (VLANs), and most are using a standardized IEEE802.1q setup.

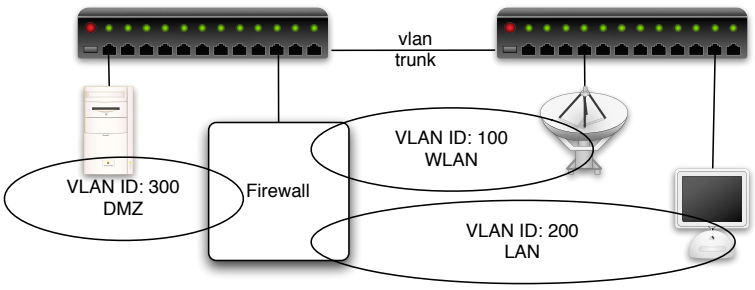


Figure 10.1: The firewall consist of multiple devices providing protection at different layers

[1] https://en.wikipedia.org/wiki/Network_segment
[2] https://en.wikipedia.org/wiki/Network_segmentation

The main thing about VLANs is that it happens on Layer 2, by adding a small L2 VLAN header to the Ethernet frames – the layer just below IP. So a packet being sent across a VLAN switch can only be in a specific VLAN. If the packet needs to be forwarded onto another segment/VLAN it must be handled by a router/firewall. VLANs are identified by a 12-bit ID field, allowing for up to 4,094 VLANs in a single infrastructure, but most networks are far from this many.

An important thing to notice is that now the securiy is also depending on the software and hardware in the switches used! This is even further extended into the Wireless access points, which need to *tag* packets before sending them to the switch for the multiple network SSIDs[3] used.

The standard is well described on Wikipedia:

> IEEE 802.1Q, often referred to as Dot1q, is the networking standard that supports virtual local area networking (VLANs) on an IEEE 802.3 Ethernet network. The standard defines a system of VLAN tagging for Ethernet frames and the accompanying procedures to be used by bridges and switches in handling such frames. The standard also contains provisions for a quality-of-service prioritization scheme commonly known as IEEE 802.1p and defines the Generic Attribute Registration Protocol.
>
> Source: https://en.wikipedia.org/wiki/IEEE_802.1Q

Using the method we can extend it to the Wi-Fi access points, as shown in the figure above, and some common isolated segments would be:

- Guest network for cabled and Wi-Fi clients
- Client networks can be isolated per department, floor or building
- Server networks for multiple purposes – development, staging, testing, production, manufacturing, …
- Internet servers in a demilitarized zone (DMZ)[4] which would make it less likely that compromised internet server would affect the rest of the network
- Dedicated printer network
- Management network for virtualisation, one for network management, one for storage, …

[3] https://en.wikipedia.org/wiki/Service_set_(802.11_network)#SSID
[4] https://en.wikipedia.org/wiki/DMZ_(computing)

Chapter 11

Building a Dual-Stack Firewall

This chapter is an attempt to show a complete example: Building a Dual-Stack IPv4/IPv6 Firewall using OpenBSD PF.

The main point is to illustrate that *firewalls* are not only about the rule-sets and policies, but often have more roles to allow the network to function. Things like handing out addresses to clients with DHCP, providing DNS service and more is part of the *firewall role*.

Your other routers and platforms will hide a lot of this functionality, so we are looking into the engine room with this long example.

11.1 Background

This example will demonstrate how to turn an OpenBSD system into a dual-stack IPv4/IPv6 router.

This gateway config inspired from https://www.openbsd.org/faq/pf/example1.html.

In this case, a router is defined as a system that performs the following duties:

- Network Address Translation (NAT)
- Requesting IPv6 prefix from upstream provider via DHCPv6
- Handing out IPv4 addresses to clients via DHCP
- Handing out IPv6 prefixes to clients via IPv6 Router advertisement
- Doing DNS caching for the LAN

This example will use two ethernet network cards named `em0` and `em1`. Your platform might have other devices, and you can build a router using a single interface. This is called a router-on-a-stick[1].

11.2 Why OpenBSD and not Linux

We have decided to use OpenBSD as the example operating system and firewall implementation in this book as explained in section 1.3. This might not be the first choice for you, but this longer example will need multiple services, like DHCP, and these are all included in a base install of this operating system.

11.3 Automated Playbooks

This example below list the commands used for performing the tasks manually. The main process transforms a generic operating system OpenBSD into being a full-fledged but basic router and firewall for a small network. This is comparable to the router in your home.

I recommend that instead of doing this manually you would soon use a configuration management system. My choice is using Git and then Ansible as the configuration management tool, which is based on Python. This also allows easy and consistent configuration of a firewall cluster with two devices configured in a high-availability / fail-over configuration.

[1] https://en.wikipedia.org/wiki/Router_on_a_stick

11.4 Networking

Let's get started with the basic routing. We will below consider `em0` as the outside Internet facing interface and `em1` as the inside interface.

The network we will use is a private IPv4 **10.0.45.0/24** subnet for the clients.

The OpenBSD network configuration is stored in small text files named after the network interfaces, `hostname.if` files.

https://man.openbsd.org/hostname.if

Other places and files of interest are:

- `/etc/` in general has most of the operating system configuration
- `/etc/sysctl.conf` store common parameters for the operating system, like forwarding of packets – turning routing on/off
- `/etc/hostname.em0` – network configuration for a network card named `em0`, create one for each physical and virtual network card
- `/etc/hosts` – the hosts file, static IP to name *database*

Each helping daemon has its own file(s) or directories, depending on needs.

11.4.1 Enable Routing

First enable routing for both protocol families:

```
# echo 'net.inet.ip.forwarding=1' >> /etc/sysctl.conf
# echo 'net.inet6.ip6.forwarding=1' >> /etc/sysctl.conf
```

Configure the external interface – assuming an upstream at a home or similar situation where the external IP is dynamic:

/etc/hostname.em1

```
dhcp
```

Static configuration if needed could be like this.

/etc/hostname.em1

```
inet 192.0.2.1 255.255.255.192
# Enable IPv6 if you got it, by uncommenting the below lines:
# inet6 alias 2001:db8:0123::82EE:73FF:FEEF:6755 64
# !route -n add -inet6 default 2001:db8:0123::1
#
# tunnelbroker config if your provider does not support native IPv6:
# 192.0.2.1 would be your address
# 198.51.100.123 would be the tunnel endpoint address
#!ifconfig gif0 tunnel 192.0.2.1 198.51.100.123
#!ifconfig gif0 inet6 alias 2001:db8:1f14:d35::2 2001:db8:1f14:d35::1 prefixlen 128
#!route -n add -inet6 default 2001:db8:1f14:d35::1
```

Configure the internal interface by creating a configuration file.

/etc/hostname.em0

```
inet 192.168.1.1 255.255.255.0 192.168.1.255'
#
!route -qn add -net 10.0.0.0/8 -interface 127.0.0.1 -reject
!route -qn add -net 172.16.0.0/12 -interface 127.0.0.1 -reject
!route -qn add -net 192.168.0.0/16 -interface 127.0.0.1 -reject
```

Note: IPv6 is enabled by default, and will be kept out of this currently. See later discussion.

11.5 DHCP for IPv4

The https://man.openbsd.org/dhcpd dhcpd(8) daemon should be started at boot time to provide client machines with local IPv4 addresses. Configuration is done via the https://man.openbsd.org/dhcpd.conf dhcpd.conf(5) file.

```
# rcctl enable dhcpd
# rcctl set dhcpd flags em1 athn0
# vi /etc/dhcpd.conf
```

The following example also provides static IP reservations for a laptop and server based on their MAC addresses.

/etc/dhcpd.conf

```
subnet 192.168.1.0 netmask 255.255.255.0 {
option routers 192.168.1.1;
option domain-name-servers 192.168.1.1;
range 192.168.1.4 192.168.1.254;
host myserver {
fixed-address 192.168.1.2;
hardware ethernet 00:00:00:00:00:00;
}
host mylaptop {
fixed-address 192.168.1.3;
hardware ethernet 11:11:11:11:11:11;
}
}
subnet 192.168.2.0 netmask 255.255.255.0 {
option routers 192.168.2.1;
option domain-name-servers 192.168.2.1;
range 192.168.2.2 192.168.2.254;
}
```

Any RFC1918 address space may be specified here. The `domain-name-servers` line in this example specifies a local DNS server that will be configured in a later section.

11.6 Router Advertisement Daemon (RAD) for IPv6

The [https://man.openbsd.org/rad] rad(8 daemon should be started at boot time to provide client machines with IPv6 prefixes for autoconfiguration. Configuration is done via the [https://man.openbsd.org/rad.conf] rad.conf(5) file.

```
# rcctl enable rad
# vi /etc/rad.conf
```

The following example is the most simple configuration

/etc/rad.conf

```
interface em1
```

11.7 Request IPv6 Prefix using DHCPv6

Many internet providers have begun to use IPv6 prefix delegation. This works by having the router request prefixes which are then added to interfaces. The OpenBSD rad daemon will automatically pick up the prefix, and start advertising the right prefix to clients.

You can read more about this process at https://en.wikipedia.org/wiki/Prefix_delegation Prefix delegation wikipedia.

The https://man.openbsd.org/rad rad(8) daemon should be started at boot time to provide client machines with IPv6 prefixes for autoconfiguration. Configuration is done via the https://man.openbsd.org/dhcpcd.conf dhcpcd.conf(5) file.

```
# pkg_add dhcpcd
# rcctl enable dhcpcd
# vi /etc/dhcpcd.conf
```

Note the package is a client daemon, dhcp client daemon – dhcpcd, while the regular DHCP server is dhcpd.

The following example is from a running configuration

/etc/dhcpcd.conf

```
ipv6only
noipv6rs
duid
persistent
option rapid_commit
#require dhcp_server_identifier
script ``''
allowinterfaces em0
interface em0
ipv6rs
iaid 1
ia_pd 1/::/48 em1/0
```

This example specifically requests a /48 and you may have to tweak some settings.

Example output using debug:

```
# dhcpcd -d
dhcpcd-9.3.1 starting
chrooting as _dhcpcd to /var/empty
sandbox: pledge
spawned privileged actioneer on PID 13087
spawned network proxy on PID 36347
spawned controller proxy on PID 23610
DUID 00:01:02:03:04:05:06:07:08:09:00:00:00:00:00:00:00:00
em0: IAID 00:00:00:01
em0: delaying IPv6 router solicitation for 0.9 seconds
em0: reading lease: /var/db/dhcpcd/em0.lease6
em0: soliciting a DHCPv6 lease
em0: delaying SOLICIT6 (xid 0x46fe10), next in 1.0 seconds
em1: activating for delegation
em1: IAID 06:07:08:09
em0: soliciting an IPv6 router
em0: sending Router Solicitation
em0: Router Advertisement from fe80::fe33:42ff:fe20:1234
em0: adding default route via fe80::fe33:42ff:fe20:1234
em0: Router Advertisement from fe80::e65d:37ff:30:1234
em0: broadcasting SOLICIT6 (xid 0x46fe10), next in 1.1 seconds
em0: ADV 2001:db8:12f0::/48 from fe80::fe33:42ff:fe20:1234
em0: broadcasting REQUEST6 (xid 0x905f73), next in 1.0 seconds
em0: wrong xid 0x46fe10 (expecting 0x905f73) from fe80::fe33:42ff:fe20:1234
```

```
em0: REPLY6 received from fe80::e65d:37ff:30:1234
em0: renew in 3600, rebind in 7200, expire in 7200 seconds
lo0: adding reject route to 2001:db8:12f0::/48 via ::1
em0: writing lease: /var/db/dhcpcd/em0.lease6
em0: delegated prefix 2001:db8:12f0::/48
em1: adding address 2001:db8:12f0::1/64
em1: pltime 3600 seconds, vltime 7200 seconds
em1: waiting for DHCPv6 DAD to complete
em1: changing route to 2001:db8:12f0::/64
forked to background, child pid 36221
```

If all goes well your internal interface will have IPv6 configured, and clients will soon pickup a prefix and configure IPv6.

11.8 Firewall

OpenBSD's PF firewall is configured via the file `/etc/pf.conf` which is described in the manual https://man.openbsd.org/pf.conf.

It's highly recommended to become familiar with it, and PF in general, before copying this example.

A gateway configuration might look like the listing shown.

```
outside = "em1"
inside = "em0"
table <martians> { 0.0.0.0/8 10.0.0.0/8 127.0.0.0/8 169.254.0.0/16 \
                   172.16.0.0/12 192.0.0.0/24 192.0.2.0/24 224.0.0.0/3 \
                   10.0.45.0/16 198.18.0.0/15 198.51.100.0/24        \
                   203.0.113.0/24 }
set block-policy drop
set loginterface egress
set skip on lo0
match in all scrub (no-df random-id max-mss 1440)
match out on egress inet from !(egress:network) to any nat-to (egress:0)
antispoof quick for { egress $inside }
block in quick on egress from <martians> to any
block return out quick on egress from any to <martians>
block all
pass out quick inet
pass out quick inet6
pass in on { $inside } inet
pass in on { $insde } inet6
```

The ruleset's various sections will now be explained in detail below.

```
outside = "em1"
inside = "em0"
```

The wired interfaces are named for the LAN and defined with macros, used to make overall maintenance easier. Macros can be referenced throughout the ruleset after being defined.

```
table <martians> { 0.0.0.0/8 10.0.0.0/8 127.0.0.0/8 169.254.0.0/16 \
                   172.16.0.0/12 192.0.0.0/24 192.0.2.0/24 224.0.0.0/3 \
                   10.0.45.0/16 198.18.0.0/15 198.51.100.0/24        \
                   203.0.113.0/24 }
```

This is a table of non-routable private addresses that will be used later.

```
set block-policy drop
set loginterface egress
set skip on lo0
```

PF allows certain options to be set at runtime. The `block-policy` decides whether rejected packets should return a TCP RST or be silently dropped.

The `loginterface` specifies which interface should have packet and byte statistics collection enabled. These statistics can be viewed with the `pfctl -si` command.

In this case, the `egress` group is being used rather than a specific interface name. By doing so, the interface holding the default route (`em1`) will be chosen automatically.

Finally, `skip` allows a given interface to be omitted from packet processing. The firewall will ignore traffic on the https://man.openbsd.org/lo lo(4) loopback interface.

```
match in all scrub (no-df random-id max-mss 1440)
match out on egress inet from !(egress:network) to any nat-to (egress:0)
```

The `match` rules used here accomplish two things: normalizing incoming packets and performing network address translation, with the `egress` interface between the LAN and the public internet. For a more detailed explanation of `match` rules and their different options, refer to the pf.conf(5) manual.

```
antispoof quick for { egress $inside }
block in quick on egress from <martians> to any
block return out quick on egress from any to <martians>
```

The antispoof keyword provides some protection from packets with spoofed source addresses. Incoming packets should be dropped if they appear to be from the list of unroutable addresses defined earlier. Such packets were likely sent due to misconfiguration, or possibly as part of a spoofing attack. Similarly, clients should not attempt to connect to such addresses. The “return” action is specified to prevent annoying timeouts for users.

Note that this can cause problems if the router itself is also behind NAT.

```
block all
```

The firewall will set a “default deny” policy for all traffic, meaning that only incoming and outgoing connections which have been explicitly put in the ruleset will be allowed.

```
pass out quick inet
pass out quick inet6
```

Allow outgoing IPv4 and IPv6 traffic from both the gateway itself and the LAN clients.

```
  pass in on { $inside } inet
  pass in on { $inside } inet6
```

Allow internal LAN traffic.

11.9 About IPv6 Neighbor Discovery Protocol (NDP)

There is a difference between IPv4 and IPv6 in handling link layer addresses. This is handled in IPv4 using ARP and the firewall is never involved in this process. With IPv6 this is different, since they are handled by the Neighbor Discovery Protocol.

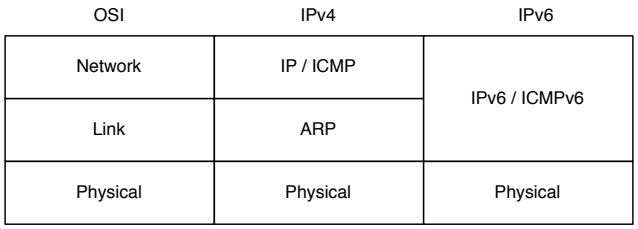


Figure 11.1: IPv4 uses ARP, while IPv6 uses NDP

The conclusion is that dependent on the firewall implementation you may need to add additional rules for ICMPv6 to work correctly. Specifically for OpenBSD PF we need to explicitly have to add rules to allow Neighbor Discovery Protocol (NDP).

The most specific configuration would be:

```
# allow ICMPv6 for NDP
pass in inet6 proto ipv6-icmp all icmp6-type neighbradv keep state
pass out inet6 proto ipv6-icmp all icmp6-type neighbrsol keep state
```

```
# server with configured IP address and router advertisement daemon running
pass out inet6 proto ipv6-icmp all icmp6-type routeradv keep state

# client which uses autoconfiguration would use this also
pass in inet6 proto ipv6-icmp all icmp6-type routeradv keep state
pass out inet6 proto ipv6-icmp all icmp6-type routersol keep state
```

but during testing it may be easier to allow any, or all IPv6:

```
pass in inet6 all
pass out inet6 all
pass in proto ipv6 all
pass out proto ipv6 all
pass in quick proto icmp6 all
pass out quick proto icmp6 all
```

You can verify NDP using the ndp(8) command

```
sakura# ndp -an
Neighbor                             Linklayer Address   Netif Expire    S Flags
2001:678:9ec:8::1                    00:0c:29:80:e8:79     em1 34s       R R
2001:678:9ec:8::53                   80:ee:73:f1:f3:d9     em1 permanent R l
fe80::20c:2908:3480:e879%em1         00:0c:29:80:e8:79     em1 23h33m41s S R
fe80::82ee:73ff:fef1:f3d9%em1        80:ee:73:f1:f3:d9     em1 permanent R l
```

This show a few IPv6 addresses are resolved to linklayer addresses, which are used when producing Ethernet packets for these destinations.

11.10 Running a Dual-Stack Web Server

If you have a webserver listening on both IPv4 and IPv6 you can add rules similar to these. The line with rdr-to can be left out if the server is not on private address.

```
table <webservers> { 2001:db8:3e4:72::1:2 192.168.1.2 }
...
pass in on egress inet proto tcp from any to (egress) port { 80 443 } rdr-to 192.168.1.2
pass in on egress proto tcp to <webserver> port { 80 443 }
```

11.11 Domain Name System (DNS)

While a DNS cache is not required for a gateway system, it is a common addition to one. When clients issue a DNS query, they'll first hit the unbound(8) cache. If it doesn't have the answer, it goes out to the upstream resolver. Results are then fed to the client and cached for a period of time, making future lookups of the same address quicker.

```
# vi /var/unbound/etc/unbound.conf
# rcctl enable unbound
```

Something like this should work for most setups:

/var/unbound/etc/unbound.conf

```
server:
interface: 192.168.1.1
interface: 192.168.2.1
interface: 127.0.0.1
access-control: 192.168.1.0/24 allow
access-control: 192.168.2.0/24 allow
do-not-query-localhost: no
hide-identity: yes
hide-version: yes
# Preferred not to use below, unless DNS queries are blocked
```

```
#forward-zone:
# name: ``.''
# forward-addr: 192.0.2.53 # IP of the upstream resolver
```

Further configuration options can be found in [https://man.openbsd.org/unbound.conf] unbound.conf(5).

If the router should also use the caching resolver, its `/etc/resolv.conf` file should contain `nameserver 127.0.0.1`.

Note: the clients should use the IPv4 addresses of the router for DNS, or IPv6 should be added to the access-control, possibly using the upstream provider /32 prefix.

Once the changes are in place, reboot the system.

Supplementary Material

11.12 Acronyms

The list below is only a short list of acronyms that I have used. For a more complete glossary the interested reader should consult one of the classic books about network security or perhaps *Internet Security Glossary* [@RFC4949].title. @RFC4949

AES American Encryption Standard
API Application Program Interface
ARP Address Resolution Protocol
AS Autonomous System
BCP Best Current Practice, an RFC sub-series
BGP Border Gateway Protocol
CIDR Classless Inter-Domain Routing
DHCP Dynamic Host Configuration Protocol
DMZ Demilitarized Zone, an isolated network segment typically used for Internet servers
DNS Domain Name System, host name lookup, reverse lookup from IP address
FTP File Transfer Protocol Activities Board)
HTTP Hyper Text Transfer Protocol – clear text web protocol
HTTPS Hyper Text Transfer Protocol Secure – TLS encrypted secure version of HTTP
IANA Internet Assigned Numbers Authority https://www.iana.org/
ICMP Internet Control Message Protocol
ICMPv6 Internet Control Message Protocol, version 6
IEEE Institute of Electrical and Electronics Engineers, Inc.
IETF Internet Engineering Task Force
IPsec IP security – packet encrypting protocols, used for VPN
IMAP Internet Message Access Protocol
IP Internet Protocol
IPv4 Internet Protocol version 4
IPv6 Internet Protocol version 6
ISO International Organization for Standardization
LAN Local Area Network
MAC Medium Access Control
MTU Maximum Transmission Unit
NAT Network Address Translation
NDP,ND Neighbor Discovery Protocol
NTP Network Time Protocol, a more simple version exist Simple Network Time Protocol (SNTP)
OSI Open Systems Interconnect model, developed by ISO
RFC Request For Comments – IETF standard documents
RIP Routing Information Protocol – an old routing protocol
RPF Reverse Path Forwarding
SMTP Simple Mail Transfer Protocol
SNMP Simple Network Management Protocol
SNTP Simple Network Time Protocol

SSH Secure Shell, remote login protocol
SSL Secure Socket Layer, see TLS
TCP Transmission control Protocol, connection-oriented protocol
TLS Transport Layer Security
UDP User Datagram Protocol, connection-less protocol
WAF Web Application Firewall
WAN Wide Area Network

For supplementary material accompanying this book, please visit https://codeberg.org/kramse

Afterword

11.13 Typos, errors and changes

If you notice typos or other issues, feel free to open an issue on Codeberg or submit a pull request.

11.14 Typesetting

The entire book is written in R Markdown and using the **bookdown**[2] package to turn a collection of markdown documents into a coherent whole. The book's source code is available on Git and hosted with Codeberg.org[3]. Underneath Pandoc[4] and especially LaTeX with a plethora of packages and classes from others are used. I installed most of this as part of the TeX Live distribution[5].
I am very grateful for the people making it possible to write books by providing quality tools!

11.15 Donate to OpenBSD

I wrote this book to cover a need, but having a more vendor neutral resource for firewalls in the OpenBSD operating system made it easier. My small donations do not express how much I love OpenBSD.
The OpenBSD project develop core technologies for the internet, such as OpenSSH – and you should really consider donating to the development. Donations and funding for OpenBSD and related products can be made through The OpenBSD Foundation:
https://www.openbsdfoundation.org/

[2] https://bookdown.org/
[3] https://codeberg.org
[4] https://pandoc.org/
[5] https://www.tug.org/texlive/

Bibliography

Cheswick, William R. 1990. *The Design of a Secure Internet Gateway.* Anaheim, CA. http://www.cheswick.com/ches/papers/gateway.ps.

Cheswick, William R., and Steven M. Bellovin. 1994. *Firewalls and Internet Security; Repelling the Wily Hacker.* Reading, MA: Addison-Wesley. http://www.wilyhacker.com/.

Cheswick, William R., Steven M. Bellovin, and Aviel D. Rubin. 2003. *Firewalls and Internet Security; Repelling the Wily Hacker. Second Edition.* Reading, MA: Addison-Wesley. http://www.wilyhacker.com/.

Kent, S., and R. Atkinson. 1998. "Security Architecture for the Internet Protocol." RFC 2401. RFC Editor; Internet Requests for Comments; RFC Editor. https://datatracker.ietf.org/doc/html/rfc2401.

Lyles, J. Bryan, and Christoph L. Schuba. 1996. "A Reference Model for Firewall Technology and Its Implications for Connection Signaling." In. Reprinted as Department of Computer Sciences, Purdue University. https://www.cerias.purdue.edu/assets/pdf/bibtex_archive/96-07.pdf.

Matt Bishop. 2019. *Computer Security: Art and Science, 2nd Edition.* Addison-Wesley Professional.

Postel, Jon. 1981. "Internet Protocol." STD 5. RFC Editor; Internet Requests for Comments; RFC Editor. https://datatracker.ietf.org/doc/html/rfc791.

Rekhter, Yakov, Robert G. Moskowitz, Daniel Karrenberg, Geert Jan de Groot, and Eliot Lear. 1996. "Address Allocation for Private Internets." BCP 5. RFC Editor; Internet Requests for Comments; RFC Editor. https://datatracker.ietf.org/doc/html/rfc1918.

Saltzer, Jerome H., and Michael D. Schroeder. 1975. *The Protection of Information in Computer Systems.*

Sanders, Chris. 2017. *Practical Packet Analysis, 3rd Edition.* No Starch Press.

Sanders, Chris, and Jason Smith. 2014. *Applied Network Security Monitoring.* Syngress.

Shirey, R. 2007. "Internet Security Glossary, Version 2." RFC 4949. RFC Editor; Internet Requests for Comments; RFC Editor. https://datatracker.ietf.org/doc/html/rfc4949.

Steven M. Bellovin. 1989. "Security Problems in the TCP/IP Protocol Suite." *Computer Communication Review* 19 (2): 32–48. https://www.cs.columbia.edu/~smb/papers/ipext.pdf.

———. 2004. "A Look Back at 'Security Problems in the TCP/IP Protocol Suite'." In *Annual Computer Security Applications Conference.* https://www.cs.columbia.edu/~smb/papers/acsac-ipext.pdf.