Firewalls 101

University Information Security

Office of Information Technology

University of Minnesota
Driven to Discover®

The current primary maintainer is University Information Security <`security@umn.edu`>.

Initial release: February 2019 (Alan Amesbury)

Contents

List of Tables

List of Figures

Preface

In July 2018 University Information Security (UIS) conducted an informal survey to gauge interest in basic, generic firewall training at the University of Minnesota (UMN). The response rate was unexpectedly high and indicated strong desire to know more about not just firewalls, but also interest in basic network training. One of the outcomes of the survey is this book.

This book is intended to provide:

- a basic IP networking overview
- descriptions of common firewall types and their characteristics
- guidance on how to develop meaningful firewall policies

It is *not* intended to substitute for vendor-specific documentation, e.g., you won't find detailed descriptions of how Linux's IPTables works or how how to configure a Fortigate.

This book is broken down into major chapters with the intent that you can skip sections covering material you already know.

There are a few typographical conventions used in this book. *Emphasized text* is used to highlight specific terms as they're introduced. `Fixed-width fonts` indicate things like addresses (e.g., `128.101.101.1`), logical operators, protocol types, commands (e.g., `PING.EXE`), or other special items.

UMN-specific sidebars appear in boxes like this. These include notes or tips that are specific to UMN.

The book is typeset in LaTeX, and is available in raw source form from UMN's Github. A PDF built on a fairly current version of the source will also be included in the repository, although it may not be updated in perfect synchronization with the book matter. Please let `security@umn.edu` know about any errors you find in this book, or use Github's features to submit an issue or pull request for correction.

"Many thanks to Tier 2 Voice & Data, AHC, and LATIS for being willing guinea pigs in terms of reviewing this document and providing feedback on some of the accompanying matter! I think the results are all the much better for it. Also, thanks to my proofreading riding buddy, Cat, who thoroughly eviscerated early versions." – *Alan Amesbury, primary author*

Chapter 1

Network introduction

> The Internet lives where anyone can access it.
>
> *Vint Cerf*

1.1 Introductory overview of networks

Information networks are everywhere and exist in many forms. They may be simple or complex. They may be implemented using a variety of techniques and technology. They may be contained within a small area or span the globe, or beyond. However, one thing they all have in common is their use for transporting information.

1.1.1 General networks

> *network*
> 4a. Any netlike or complex system or collection of interrelated things, as topographical features, lines of transportation, or telecommunications routes (esp. telephone lines).
> 4d. Computing. A system of interconnected computers. Frequently attributive.
>
> *"network, n. and adj.". OED Online. July 2018. Oxford University Press. http://www.oed.com/view/Entry/126342?rskey=ETyfe8&result=1&isAdvanced=false (accessed October 22, 2018)*

One large network used daily by millions of people in the United States is the U.S. postal system. It processes and delivers hundreds of millions of pieces of mail each day.[1] In spite of its size and complexity, people are able to use it with little to no thought. Someone sending a package merely

[1] `https://facts.usps.com/size-and-scope/`

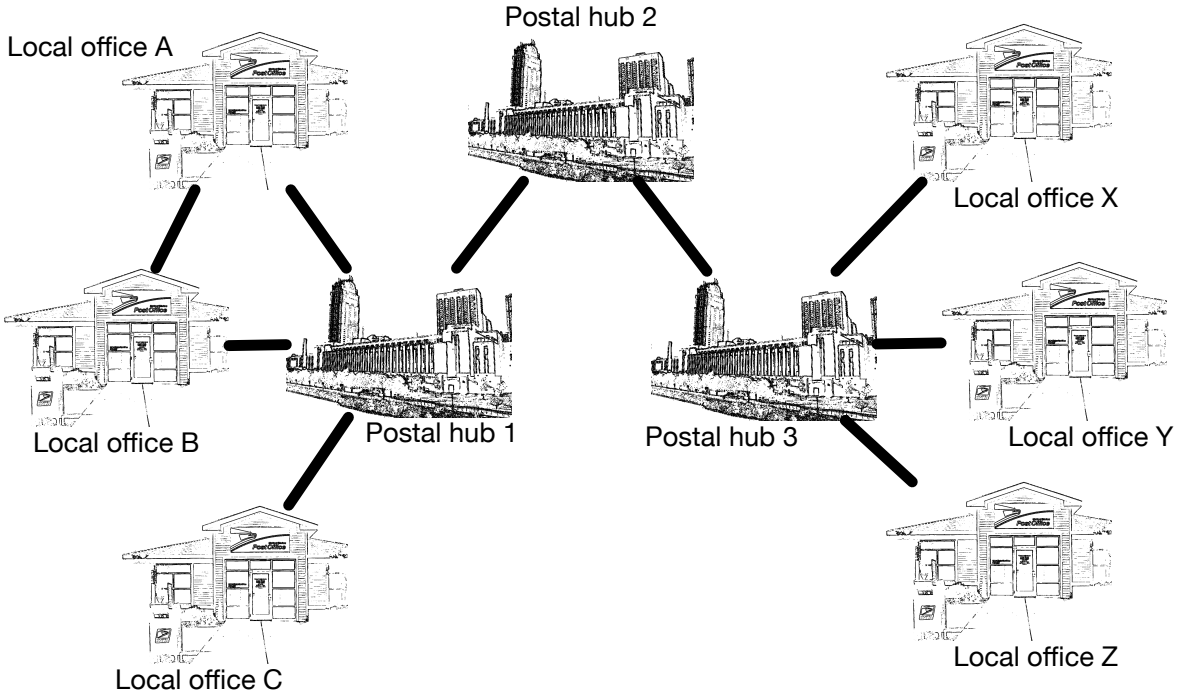

Figure 1.1: A sample postal network

has to pack it in a way the postal service will accept, write the address on the outside in a form the postal system can parse, pay any necessary transport fees, and hand the package over for delivery. Packaging can vary depending on what's being sent, and the postal network is able to accommodate a wide variety of packaging methods. The user of the network doesn't have to consider the details: locating the destination, handling unexpected problems like weather or transportation delays, ultimately delivering the package, and other related tasks are all handled by the network.

There is some specialization within the postal network. Local post offices mostly handle deliveries within their local areas. For deliveries that aren't to the local area, they hand things off to other post offices. The local post office may know of other local post offices and have some understanding of the areas they serve, so it may be able to hand things off directly to the appropriate neighboring post office for delivery. More often, it hands things off to a postal hub for additional handling. The hub, which may or may not also perform local deliveries in its own area, maintains a list of which local post offices deliver to which local areas, and routes the package to the appropriate local post office for final delivery. It also knows of other hubs and, very generally, has some idea of their areas of responsibility, and can send packages destined for delivery to another hub, based on its list of which hubs cover which general regions.

Figure 1.1 shows a sample postal network. (Not shown are the homes, businesses, and other postal customers where mail originates or to which it's delivered.) Each local office can deliver to addresses in its own area, or send nonlocal deliveries through whatever connections exist to other offices or hubs. Local office A has a direct connection to local office B, so a package originating in A's area destined for locations in B's area can be handed off directly to B and vice versa. Deliveries

originating in A's area destined for areas other than those handled by A or B are forwarded to postal hub 1. For example, a letter originating in A's area bound for a destination address in C's area is forwarded to hub 1, which forwards it to C, which then delivers it.

Local offices A, B, and C are all served by the same hub. Local offices X, Y, and Z are served by a different hub. Neither of these hubs can communicate with the others directly. Mail originating from an addresss in X's area bound for an address in C's area will first be forwarded to hub 3, because that's the only path local office X has for deliveries outside its own area. Hub 3 has no idea how to reach C, so follows its own default action of forwarding the message to hub 2. Hub 2 knows local offices A, B, and C are reachable via hub 1, so it forwards the message to hub 1. Hub 1 forwards the message to C, which performs final delivery.

This example illustrates several things:

- It isn't necessary for every node to know the specifics of the entire network.
  - Hub 2 needs to have an overall awareness of which major hubs serve which regions, but not the specifics of how messages are delivered.
  - Local offices need to be able to forward mail destined for nonlocal addresses to someone who can do local delivery, or knows how to forward things to the network.
  - Hubs 1 and 3 need to know which local offices they serve, but none of the specifics of how those local offices perform delivery.
- Mail always takes the most specific route available. Mail from A's area to B doesn't pass through the hub, because the direct connection is faster. Similarly, a package coming into hub 1 that destined for A's delivery area doesn't get handed off to B even though B and A talk directly with each other.
- Nodes nearer to local delivery areas, or the *edges* of the network, deliver on a path leading to postal hub 2 by default, i.e., this is a *default route*.
- In the absence of any other controls, a package with nonexistent destination addresses will always get passed to hub 2. If hub 2 is unable to match an address to a specific delivery office, an error occurs, triggering error-handling procedures.[2,3]
- The methods used to deliver a package locally are independent of each other and can be designed to best serve the local area. For example, if local office A's delivery area consists primarily of islands in swampland and C's delivery area is mostly modern office buildings, A's delivery fleet can consist of swamp boats and C's delivery fleet can consist of cargo vans. Neither needs to know about this, so long as delivery can occur.
- The methods used to forward a package between nodes are also independent of each other. If A and hub 1 are located at opposite ends of a canal, but C and hub 1 are linked by highway, the use of a barge between A and hub 1 has no effect on the use of a truck between C and hub 1.

Generally speaking, the postal system is content-agnostic[4], i.e., it doesn't care what's being transported as long as the packaging conforms to certain standards and the outside labeling is legible. The postal service usually doesn't look at package contents, except during some error conditions (e.g., a package breaks open in transit).

Some organizations, and most countries, implement their own internal postal network to provide internal mail delivery. Internal mail is delivered per the procedures defined for the internal postal

[2] `https://www.uspsoig.gov/blog/undeliverable-addressed-costs-more-you-think`

[3] `https://postalpro.usps.com/storages/2017-08/UAAMailProcessFlow.pdf`

[4] Yes, some things are prohibited, e.g., explosives or caustic chemicals, but these are atypical items.

network, but delivery of mail to destinations outside that network are handed off to another postal network for delivery. The internal implementation details of these networks are unimportant to the postal system overall, as long as a standard method for handing off mail between them exists. Similarly, knowledge of the details of postal networks in other countries isn't needed so long as there's a standard way to hand off mail between them and our postal network.

With a few exceptions, postal networks provide best-effort delivery without guarantee of delivery. If the postal network detects an exception to normal delivery, e.g., the delivery address can't be found, mail is being forwarded, a package is damaged enroute and the destination address is no longer readable, or a "vacation hold" is in effect, it may take additional steps to address the exception. These vary depending on the network.

The postal network usually delivers to addresses, not specific recipients. Once something has been delivered to the address, the final delivery is handled by people at the address, not the postal network.

UMN'S OWN NETWORK can be thought of as a microcosm of the Internet in general, and many of the concepts pertaining to the Internet at large will also apply to at least some degree to UMN's network. Internet Protocol (IP) networking is IP networking, no matter whose network it is.

1.1.2 IP networks

Many electronic networks behave very similarly to postal networks, particularly IP networks. Like the postal network, delivery is to a destination address. Local area networks (LANs) know how to do deliveries within their local area, but hand off deliveries bound for destinations outside their bailiwick. Local networks do not need to keep track of how every network reaches every other network, only how to deliver locally or hand off nonlocal deliveries so they can be transported to their intended destination. Wide area networks (WANs), operated by Internet service providers (ISPs), handle delivery over larger distances.

IP networks are content-agnostic, and can transport almost any kind of data encapsulated in the correct format and addressed correctly. IP networks do *not* perform delivery to specific programs or users; they deliver data to addresses. In the same way that the postal service does not normally guarantee the order in which multiple packages arrive, or even that they arrive at all, IP networks do best-effort delivery with no guarantees regarding order of arrival.

The network itself is concerned with delivery to addresses, but it's often useful in modern networks to consider other aspects of network data (the source address, the IP protocol, the payload, etc.) when making decisions to transport data. It's devices like firewalls that provide these additional controls.

The "packaging" used in IP networks is called a *packet*. Packets are strings of payload data (the data being transported) and any accompanying control information needed to get that payload to its destination. In electronic networks, packets usually consist of a *header* placed before the payload, the payload itself, and sometimes a packet *trailer*.

For IP packets there is no trailer. The packet header contains addressing, information about the type of data being transported, and some other basic information. The IP payload immediately follows the header and may begin with an IP protocol header or consist of just data. Although IP

networks are not the only type of electronic network, they're ubiquitous and serve the bulk of the data delivery needs between UMN and the outside world, so we will focus on them.

1.2 IP network attributes

IP networks have several attributes that are helpful to know and understand, particularly speeds and the behavior of common network devices. First and foremost, many modern LANs work through a method called collision detection[5]. It's similar to the way conversations work in a group of people. Someone waiting to speak will first listen to see if there's already someone talking, and will wait if that's the case. If it's quiet, it's OK to start talking. Sometimes two or more people start talking at the same time, so they stop and eventually one will begin speaking again. If someone is talking and someone else talks over them, messages can get garbled and have to be repeated. Wired and wireless networks both work in this same basic way, where multiple "conversations" between devices can happen, but not always at the same time.

The vast majority of networks at UMN are Ethernet or Wi-Fi-compatible wireless, and physical connectivity is primarily twisted-pair Ethernet and fiberoptic. Other types and modes of connectivity (e.g., token ring, thinnet, Fibrechannel, non-Wi-Fi wireless, or X.25) are not discussed.

1.2.1 Common network devices

The term "network device" covers a vast array of devices of differing types and capabilities, but there are a few types that are extremely common. In wired networking, there are switches, routers, firewalls, gateways, and hubs. Wi-Fi networking devices include gateways, routers, and access points. While some of these may perform vaguely similar functions, they are distinct and operate in different ways.

This section refers to network layers. These are covered in more detail in section 1.3, but it should not be necessary to understand exactly what they are for this section.

Wired LAN connectivity

LAN connectivity is primarily about layer two communications, the network connectivity usually contained within a local site

Hubs are some of the oldest types of network equipment you might encounter, and they are increasingly rare. They have a minimum of two physical network ports, as their purpose is to transfer network traffic between local, connected hosts. When a packet arrives on one port, it's copied and transmitted out all the other ports. This most closely resembles the example of a roomful of people talking, as everyone connected to the hub can hear every conversation. It's up to those connected to the hub ("in the room") to politely not listen to conversations in which they're not participating, a potentially serious security flaw. Also, "chatty" devices, i.e., ones that frequently

[5]Carrier Sense Multiple Access with Collision Detection, or CSMA/CD. See `https://en.wikipedia.org/wiki/Ethernet` for more information.

transmit data without an opportunity for another device to step in, will likely cause overall network performance to suffer.

Switches are similar to hubs in that they are used to connect local hosts together. Unlike hubs, switches learn the local media access control (MAC) addresses[6] assigned to each connected device's physical network adapter. When a device attempts to talk to another device on the LAN, the switch passes that communication to only the intended recipient by sending it out the port associated with that device. Switches continually update their knowledge of which MAC is attached to which physical port. This is analogous to a people in separate rooms conversing with each other through an intercom system run by an operator. They don't have to consider who else is already talking; they just push a button and talk. The intercom operator recognizes who they're talking to and sends the audio to the correct recipient. Networks using switches tend to be more (sometimes much more) performant than similar networks using hubs. However, under certain conditions (e.g., if the "operator" in the switch is overwhelmed), switches can "fail open" and start behaving just like a hub. Once the switch is no longer overwhelmed, it should resume normal switched operation.

Wi-Fi LAN connectivity

Wi-Fi[7] communication is analogous to people standing in an open field and shouting at each other. Any time one person is talking (sending), everyone needs to be quiet so the listener can hear the message (receiving). Encryption may make it difficult for eavesdroppers to understand the conversation, but the conversation itself can be heard by anyone in earshot.

A Wi-Fi *access point* transforms Wi-Fi network traffic to a wired form and vice versa. This is a form of *media conversion*, as data is being between a wireless medium and a wired medium, and is the point where the wired network is accessed by Wi-Fi devices. Access points are sometimes called *wireless gateways*, but that term is imprecise and can mean several different things depending on context and interpretation. Some access points include a built-in switch to permit connection of more than one wired device. Still others, particularly models designed for home or small office networks, may include routing or firewall functionality.

Devices containing the functionality of an access point, switch, and other capabilities are sometimes called *wireless routers*. Like "wireless gateway," the term is imprecise, and you should include additional context when using the term.

WAN connectivity

WAN connectivity is all about communications between networks, i.e., usually across physically distinct locales. This tends to happen at layer three.

Routers are devices that behave similarly to switches, but route traffic outside of LANs. They participate in LAN communication just like other devices, but look at layer three addresses to determine where to send packets destined for devices not on the LAN. Destinations may be on another LAN to which the router is also connected, or may be to a network to which the router is not directly connected. In the latter case, the router check to see if it has a specific route for the destination address and, if so, will forward traffic appropriately. Routers usually have a fallback (default) route to which they forward traffic for which they lack a more specific route. The term "router" is sometimes used to describe home-network devices which allow connectivity from inside

[6] MAC addresses are assigned by the hardware vendor. These can be manually changed on some devices, but doing so is not the norm.

[7] https://www.wi-fi.org/

the home to sites outside. This device usually performs several tasks, only one of which is actually "routing."

Firewalls may act like a router or switch, depending on how they're connected, but with the singular purpose of providing access controls to one or more of the networks to which they're connected. There are several types, and they are described in considerable detail in chapter 2.

A *gateway* is a device that allows access from one network to another. On its own, the term is non-specific, so its use is generally discouraged without additional qualification. For example *default gateway* usually refers to the default route configured on a device. A *home router* is also a type of gateway because it provides access from the home's wired or wireless network to the network infrastructure (e.g., to the coaxial cable network used by some Internet service providers). Additionally confusing is that some gateways also provide firewall or Network Address Translation (NAT) services. Use of a more specific term is preferred.

1.2.2 Describing performance

Network *throughput* or *bandwidth*, the rate at which data can be transferred, is described in several ways, but *bits per second (bps)* and *bytes per second (Bps)* are two of the most common. Network engineers and network equipment tend to use bps because networks transfer data one bit at a time. Users of networks, i.e., not the ones who maintain the networks, tend to think of transfer rates in terms of Bps because they usually think of the data being transmitted in terms of bytes. The relationship between bps and Bps is straightforward. For a given network, the rate in bps will be about 8 times its speed measured in Bps. There may be some small variation due to network *overhead* (like additional control data that transits the network along with the data it transports, congestion, etc.), but it's close enough for most purposes. Metric prefixes[8] are often prepended to bps and Bps to indicate magnitude, e.g., 5 Mbps for 5 megabit.

There is a difference between theoretical maximum speed and true network throughput, and the latter can be much lower than the former. Wired Ethernet networks are often rated at 10Mbps, 100Mbps, 1,000Mbps ("gigabit" or 1Gbps), and 10,000Mbps ("ten gig" or 10Gbps). The practical throughput of these networks is lower than these theoretical maximums, and a common rule of thumb is you should expect no more than about 85% of the rated throughput for wired networks. It's certainly possible to exceed that, but that should not be the expectation.

Wireless networks are rated the same way, i.e., in bps, but are more susceptible to interference from other devices. You should expect throughput to be below the rated speed, more so if you are in an area in which there are many wireless clients (e.g., a classroom, apartment building, or crowded stadium). Various wireless standards are covered by the term "Wi-Fi," and each has standard has its own operating characteristics (maximum possible throughput, susceptibility to interference, maximum effective distance, etc.). These are distinct from and unrelated to cellular-type networks, e.g., "4G," "LTE," "5G," and so on.

[8] `https://en.wikipedia.org/wiki/Metric_prefix`

NETWORK THROUGHPUT AT UMN CAN BE EXPECTED to be fairly quick for wired connectivity. The network's backbone is all 100Gbps, connectivity to most endpoints is gigabit, and datacenter-located systems are being increasingly connected at speeds faster than gigabit. Wireless throughput can be expected to be at least 60Mbps in most situations, and persistent throughput problems should be reported to `help@umn.edu`.

1.3 IP packets and layers

Conceptually, networks work in a "layered" format. The OSI model[9], mostly shown in table 1.1 (page 9), is the model we'll use. Layer 1, the physical connectivity layer, is not shown because we're not concerned about physical connectivity and electrical signaling details. Also, most firewalls don't provide layer 1 controls.

The other major network model is the TCP/IP model, also known as the Department of Defense (DoD) model. Its four-layer structure maps approximately to the "Contains" column in table 1.1, i.e., from the top to bottom they are the application layer, transport layer, network layer, and link layer.

The layers are largely independent of each other. Applications don't need to know what's going on at the transport layer or below.[10] Transport protocols like Transmission Control Protocol (TCP), User Datagram Protocol (UDP), or Generic Routing Encapsulation (GRE) are content-agnostic, and the payload they transport has no effect on them. Network protocols like IP are similarly content-agnostic. Transport protocols are completely encapsulated in the network layer as that layer's payload. The network layer can run over anything[11] providing a data link, and the method of delivery doesn't matter.

Each layer's payload contains the layer above it, e.g., the data link layer's payload contains the network layer, whose payload in turn contains the transport layer, and so on. Some protocols do span layers, e.g., ARP, but many others, e.g., Hypertext Transfer Protocol (HTTP), Domain Name System (DNS), Hypertext Transfer Protocol Secure (HTTPS), or Simple Mail Transfer Protocol (SMTP), do not.

1.3.1 IP addresses

There are two major types of IP: Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6). IPv4 addresses are 32 bits long, while IPv6 addresses are 128 bits. These versions are incompatible, but the way they're used is fairly similar. IPv4 addresses are 32 bits long, and IPv6 addresses are 128 bits long (four times the length!). Each has a distinct, commonly used way to represent them in forms that are (fairly) easily read by humans. (A brief primer on binary, hexadecimal, and conversions between those numeric bases and base 10 can be found in appendix A.)

[9] `https://en.wikipedia.org/wiki/OSI\_model`

[10] Well, usually not. There are some exceptions like the address resolution protocol (ARP).

[11] Including carrier pigeons (`https://www.ietf.org/rfc/rfc1149.txt`)

OSI Layer	Contains	Description
7 (application), 6 (presentation), 5 (session)	Application data	These layers contain the actual application data. The OSI model explicitly defines these layers, but not many applications actually distinguish between them. It is often practical to refer to them collectively as "application data" or the "application layer."
4 (transport)	IP protocol used for transport	IP protocols (e.g., TCP, UDP, or GRE) reside in this layer. For IP packet fragments that are not the first of a series of fragments, this only contains payload data without an IP protocol header. For unfragmented packets or the first fragment of a fragmented packet, it contains an IP protocol header and payload (the contents of the layers above).
3 (network)	IP packet (addressing information)	The source and destination IP addresses are found at this layer, along with other control information, e.g., time to live or packet checksum. This information is contained in an IP header, and the IP packet payload, which consists of the IP protocol layer and everything it contains, is appended to the header.
2 (data link)	Local area communications (local data link)	Communications at this layer are the local connection fabric between computers physically connected to the network. Ethernet and Wi-Fi are both extremely common. IP packets, and everything they contain, are encapsulated in data link *frames*. Frames at layer 2 are analogous to packets at layer 3.

Table 1.1: Parts of the OSI model

IPv4 and IPv6 can coexist on a single network and systems that are *dual-stacked* (support both protocols simultaneously) can use either network protocol with ease. Most modern operating systems (OSes) are dual-stacked, so devices you connect to UMN's network are likely already using IPv6 (whether you intend them to or not). UMN's network has natively supported IPv6 since at least 2008.

IP addresses are strings of bits, but addresses in binary like `10000000011001010110010101100101` (32 bits) are cumbersome for people, so we use representations that are easier to read and understand. Grouping the bits by byte and writing it in binary as `10000000.01100101.01100101.01100101` is slightly less cumbersome, but still not easy.

IPv4 addresses can be represented in pure decimal form. Some web browsers will accept a purely decimal value as input. (Try accessing `http://2253682439` for an example.) However, none of these methods are particularly easy for people to use, so someone invented a better way to encode the data.

For IPv4, addresses each group of eight bits is converted from binary (base 2) to decimal (base 10). This example address is usually represented as `128.101.101.101`. This is called the "dotted quad" format because it's four bytes separated by dots.

IPv6 addresses are four times longer, and binary representations like `00100110000001111110101000000000000000010000001001001000000000010000000000000000-0000000000000000000000000000000000000000000001010011` are even less readable to humans. Dotted representations won't work, because reading addresses like `38.07.234.0.1.2.72.1.0.0.0.0.0.0.0.83` is still cumbersome.[12]

Address	Description
`2607:EA00:0102:4801:0000:0000:0000:0053`	A complete IPv6 address
`2607:EA00:0102:4801::0053`	Same address as above, but with the three "0000" groups compacted
`2607:EA00:102:4801::53`	Same address as above, but with the three "0000" groups compacted and leading zeroes within groups removed

Table 1.2: Different forms of IPv6 addresses

IPv6 uses hexadecimal (base 16) for its written form, with eight groups of two bytes (16 bits), with two-byte groups separated by colons. Each digit represents four bits. If one or more consecutive two-byte pairs is zero, they can be compressed using the shorthand "::". This shorthand may only be used once. Leading zeroes within a two-byte group may also be dropped. Table 1.2 shows examples of this shorthand.

Because IPv4 and IPv6 are fundamentally incompatible, several transition methods were developed to bridge the gap. These include Teredo (found on Windows systems), tunnel brokers (services that allow for one protocol to be tunnelled within another), and several others.

[12] Also, nobody wants to work in something called "dotted decahex" or "dotted hekkaideca." "Dotted quad" is hard enough.

IPv6 transition technology is unnecessary on UMN's network. These features should be disabled in the absence of an explicit business need to have them enabled.

One of the key features of IPv6 is its total address space is... large. The maximum number of possible values for an IPv4 address is 2^{32} or 4,294,967,296. Not every possible value is usable for communicating over the Internet, though, as a number of blocks of addresses are reserved for various purposes. (This is covered in more detail in section 1.3.5.) IPv6 has a maximum number of possible values that's represented by four times as many bits, and each bit doubles the amount of possible space. That's 2^{128} or 340,282,366,920,938,463,463,374,607,431,768,211,456 (or about "3" followed by 38 zeroes). Like IPv4, the entire address range is not usable as some of it is reserved, but there's still a lot left over.

To put it in perspective, think of the entirety of IPv4's address space as an area measured in micrometers. It's 65,536 μm, or about 2.5 inches per side. In contrast, IPv6's address space would be 18,446,744,073,709,551,616 μm, or about 11,462,275,358 miles, on each side. NASA says the distance from the Sun to Pluto is only about 3,670,000,000 miles[13], or about a third of the distance.

IPv6 address space is big.

1.3.2 IP addresses and netmasks

In IP networks, devices connected to a network do not usually keep track of or know to reach devices on other networks. Large networks are broken apart into subcomponents, or *subnets*. Each subnet has its own network address, and each host within the subnet has its own host address. The complete IP address includes both of these components, but does not provide on its own a way to tell which part is the network's or subnet's address and which part represents a specific host.

A *subnet mask* is used to identify these parts. The mask consists of a field of bits the same length as the address (32 bits for IPv4 and 128 bits for IPv6), with zero or more consecutive bits set starting with the left-most (most significant) bit. The netmask is logically *AND*ed[14] with the address. This operation *masks* bits that are not part of the subnet's address, and the result is the base address of the subnet.

For example, suppose you have a host with IP address `128.101.238.97` and a netmask of `255.255.255.192`. To calculate the netmask, `AND` the values together. Table 1.3 shows the IP address (`128.101.238.97`) and netmask (`255.255.255.192`) in their binary forms. The result of `AND`ing these gives the network address in binary, which is translated back to its dotted quad form on the left. The *broadcast address* of the subnet is the network address with all the masked bits (the ones corresponding to zeros in the netmask) set to `1`.

Key things to remember:

- The only two things needed for a host to talk on an IP network are an IP address and a netmask. This won't allow the host to talk to hosts on a different subnet, but is enough for it to be able to talk to other hosts on the same subnet.
- A subnet mask's binary form always contain zero or more unseparated `1`s. A netmask that contains a `0` between a pair of `1`s is an error. How a device misconfigured in this way behaves is undefined.

[13] `https://www.nasa.gov/audience/foreducators/5-8/features/F_Solar_System_Scale.html`

[14] See `https://en.wikipedia.org/wiki/Boolean\_algebra` for information on boolean, or binary, logic.

Dotted quad values	Binary form
`128.101.238.97`	`10000000 01100101 11101110 01100001`
`255.255.255.192`	`11111111 11111111 11111111 11000000`
`128.101.238.64`–ANDed value (network address)	`10000000 01100101 11101110 01000000`
`128.101.238.127`–Broadcast address (network address with masked bits set to `1`)	`10000000 01100101 11101110 01111111`

Table 1.3: IP address and netmask in binary form

- All bits masked out by a netmask should be zeroed when referring to a subnet. In the example above, specifying a network address of 128.101.238.65 with a netmask of 255.255.255.192 to refer to the subnet above is not correct. How this is interpreted by various network tools varies.
- It's possible for hosts with the same network address and differing subnet masks to communicate if by chance they both happen to fall into the range defined by the most restrictive subnet mask. This is still a misconfiguration, as all hosts within the same subnet should have the same netmask.
- The highest and lowest addresses within a subnet are usually reserved. The highest address is the broadcast address for the subnet, and the lowest is considered the subnet's own address. This is why IPv4 netmasks greater than 30 bits usually don't make sense.[15]
- IP addresses and subnet masks work the same for IPv6. They're just much longer than IPv4, and they're not represented in dotted quad.
- An IPv4 netmask of `255.255.255.255` is all `1`s. This refers to a specific IP address without any additional information about the subnet to which it might belong.

Decimal value	Binary value
255	`11111111`
254	`11111110`
252	`11111100`
248	`11111000`
240	`11110000`
224	`11100000`
192	`11000000`
128	`10000000`
0	`00000000`

Table 1.4: Values common to dotted quad IPv4 netmasks

[15]There are situations that call for a 31-bit netmask in IPv4 subnets, but these are uncommon. Consult with a networking specialist if you think you're supposed to be using netmasks like these.

Table 1.4 shows the valid values for parts of an IPv4 netmask and their binary values. You may not need an equivalent for IPv6, because the netmasks you're most likely to encounter are just 32, 40, or 64 bits long. (Remember, IPv6 addresses and netmasks are 128 bits long.) Instead of being represented in the same way IPv6 addresses are, we use CIDR notation (covered in section 1.3.3).

The terms "netblock" and "subnet" are sometimes used interchangeably. Here a subnet specifically includes the addresses specified by a network address and its netmask. *Netblock* can refer to a subnet, a collection of subnets, a collection of individual addresses, or other grouping of addresses and subnets.

1.3.3 CIDR notation

CIDR	Base address	Netmask	Broadcast address
128.101.238.64/26	128.101.238.64	255.255.255.192	128.101.238.127
128.101.0.0/16	128.101.0.0	255.255.0.0	128.101.255.255
131.212.16.0/20	131.212.16.0	255.255.240.0	131.212.31.255

Table 1.5: IPv4 CIDR notation examples

Classless Inter-Domain Routing (CIDR) notation is shorthand notation for subnets and netmasks. It's comprised of the base network address, a slash, and the number of `1` bits in the netmask. This notation is used for IPv4 and IPv6. In the example in section 1.3.2, we determined the network address was `128.101.238.64`. The number of bits in the netmask, `255.255.255.192`, is 26. (See table 1.4 for hints on converting this.) This subnet is represented in CIDR notation as `128.101.238.64/26`. Table 1.5 shows several more valid examples of CIDR notation for IPv4 addresses, along with the resulting base network addresses, dotted quad netmasks, and broadcast addresses.

As mentioned in section 1.3.2, bits masked out by the netmask should usually be zero. Notation with a masked bit set may take on different meanings depending on context, tools used, and other variables. It is common to see a host and its netmask in CIDR notation, but this should only be done when it is clear you're referring to a specific host and subnet. References to the subnet itself should always have the masked bits set to zero. Table 1.6 (page 14) shows some additional CIDR notation examples and their possible meanings.

1.3.4 Default route

The *default route* in an IP network is the path taken by traffic not destined for the local subnet. The *default router* must be a host or device configured to pass traffic between subnets or networks; general-purpose computing hosts will generally not route traffic without special configuration. Special-purpose network equipment known as *routers* are designed specifically to route traffic between subnets or networks.

Like every other device in an IP network, the router must have an address of its own. The router may have any valid address within the subnet(s) it serves, but there are no other constraints within IP networking regarding what that address must be. Other devices are configured to use the router as the default delivery path for network traffic. It is possible for a subnet to have more than one router, for more than one router to act as a default route, and for a subnet to have no default route.

CIDR	**Interpretation(s)**
"`128.101.238.64/26`"	Unambiguously refers to subnet `128.101.238.64` (all masked bits are zero) with a 26-bit netmask
"`fe80:0:0:358f::/64`"	Unambiguously refers to subnet `fe80:0:0:358f::` (all masked bits are zero) with a 64-bit netmask
"host `128.101.238.97/26`"	Identified explicitly as a host, so is expected to refer to the host with IP address `128.101.238.97` on subnet `128.101.238.64/26`
"host `fe80::4:2a6f:d7e8:8fa3/64`"	Identified explicitly as a host, so is expected to refer to the host with IP address `fe80::4:2a6f:d7e8:8fa3` on subnet `fe80:0:0:4::/64`
"`128.101.138.0/17`"	Ambiguous: may refer to subnet `128.101.128.0/17`; has masked bits set, so may also refer to host `128.101.138.0` in subnet `128.101.128.0/17`
"`128.101.101.255/24`"	Ambiguous: may refer to subnet `128.101.128.0/24`; has all masked bits set, so may also refer to the broadcast address (`128.101.101.255`) on that subnet
"subnet `10.31.17.192/25`"	Error: explicitly refers to a subnet with a nonsensical netmask; can't tell if this is supposed to be `10.31.17.192/26` or `10.31.17.128/25`
"`fe80::64::/64`"	Error: incorrect notation for compressed IPv6 addresses (refer to section1.3.1 for how this works); could refer to several things, like subnet `fe80:0:64::/64`, subnet `fe80:0:0:64::/64`, or host `fe80::64` in one of several subnets

Table 1.6: More CIDR notation examples

Information about the route traffic takes after leaving the subnet is generally unavailable to hosts on a subnet, but there are ways to determine what the route taken *might* be. Tracing the route traffic takes is covered in more detail in section 4.2.

THE DEFAULT ROUTE at UMN for a subnet is usually the highest usable address within the subnet, just below the subnet's broadcast address. In the example in section 1.3.2, the default router's address is 128.101.238.126. UMN's networks usually do not have redundant default routers available, as redundancy is handled in other ways.

1.3.5 "Private" and other special addresses

Address(es)	Description	Reference
`10.0.0.0/8`, `172.16.0.0/12`, `192.168.0.0/16`	Private addresses set aside for organizations to use internally	RFC1918
`100.64.0.0/10`	Used for carrier-grade NAT and other private uses	RFC6598
`169.254.0.0/16`	IPv4's "link-local" address range and Microsoft's "auto-IP" (excluding `169.254.0.0/24` and `169.254.255.0/24`)	RFC3927
`FC00::/7`	Unique local address (ULA) (functionally limited to `FD00::/8` in the RFC)	RFC4193
`FE80::/10`	IPv6's "link-local" address range	RFC4862

Table 1.7: Common private/special addresses seen on UMN's network

There are several netblocks of addresses that do not work for general connectivity to and from the Internet, but may work internally within an organization. The best known are often referred to by the Requests for Comments (RFCs) that defines them: RFC1918.[16] These addresses are generally not usable for directly communicating outside an organization. The closest equivalent in IPv6 is the ULA netblock.[17]

[16] `https://www.ietf.org/rfc/rfc1918.txt`
[17] ULA is defined in RFC4193. See `https://tools.ietf.org/html/rfc4193` for more information.

RFC1918 addresses were historically used to prevent devices from being accessed over the Internet. Some RFC1918 addresses are routed within UMN's network and assigned to devices like printers. However, the inaccessibility of RFC1918 addresses over the Internet is only a side effect of how packets destined for RFC1918 addresses tend to not transit the Internet. Although historically this has provided some protection for UMN resources against attack over the Internet, using a firewall is the recommended and preferred way to control network access.

Private/special addresses you're likely to see in use on UMN's network and hosts are shown in table 1.7. RFC1918 addresses are commonly used for devices which are never intended to communicate with hosts on the Internet. ULA may be used similarly in the future. Link-local addresses are frequently used by hosts as part of self-configuration; IPv4 link-local addresses often show up on hosts unable to obtain a Dynamic Host Configuration Protocol (DHCP) lease. IPv6 link-local addresses are always self-assigned when a host capable of communicating over IPv6 connects to a network, *even if the network to which it's connected doesn't natively support IPv6.*

Network infrastructure uses the private range reserved for carrier-grade NAT.

1.4 The IP header

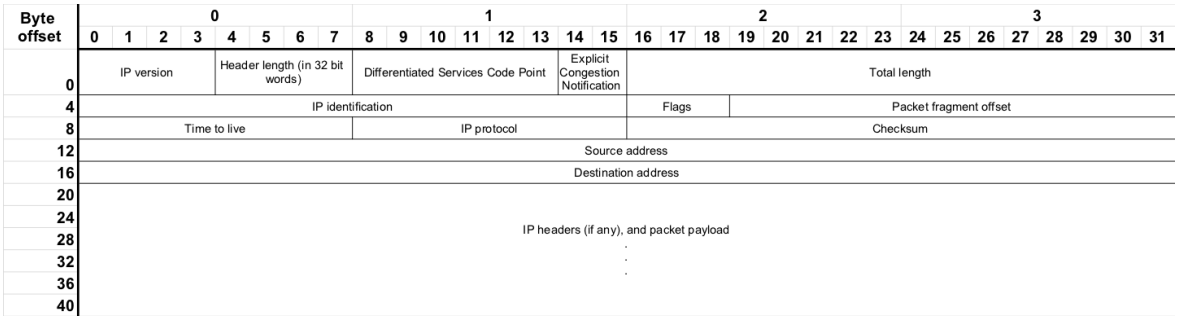

Figure 1.2: IPv4 header

IP networks are mostly content-agnostic,[18] but IP packets have a specific format. They consist of an IP packet header followed by a payload. The header format for IPv4 is often represented at a bit level as shown in figure 1.2. The offsets indicate where various components of the header can be

[18]IP networks can be configured to behave in a wide variety of ways, but some aspects of operation are common. This caveat should be kept in mind for any statements describing how an IP network behaves, because these are the rules IP networks follow... except when they don't.

found. For example, the source address starts at the 12th byte offset and is 32 bits long. The destination address, also 32 bits long, starts at the 16th byte. Other parts include a time-to-live, identification of the type of IP protocol in the payload, certain control flags, and more. Every IPv4 packet header follows this format.

For the purposes of controlling access to your network with a firewall, the IP header fields you will most often explicitly use are the:

- source address
- destination address
- IP protocol

Missing from this list are ports and the IP protocol version. Ports are not part of the IP header; they're part of the IP protocol contained in the IP packet's payload. The version of IP in use (IPv4 or IPv6) is also missing from this list because the addresses themselves are sufficient to identify the version in use. (See section 1.3.1 for details on the differences between IPv4 and IPv6 addresses.)

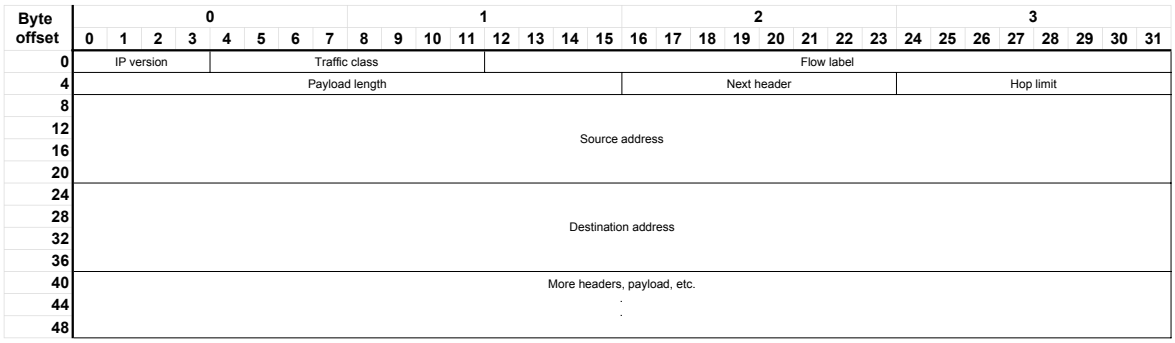

Figure 1.3: IPv6 header

The IPv6 header, shown in figure 1.3, is somewhat simpler. Instead of trying to put all options into a single header, IPv6 headers are appended to the first header, using the "next header" field to indicate where they are. For an IPv6 packet, the key fields are still the source and destination addresses, and the IP protocol. Following the IP header (or chain of headers as can happen with IPv6) is the packet *payload*.

IP packets can be fragmented in transit, but IP takes care of rebuilding fragmented packets as they're received on the destination host. In transit, however, you may see IP packets that contain a clearly identified IP protocol, or packets that contain fragments of a payload. The latter may make it difficult to identify the IP protocol. For positive identification, you'll need to see the reassembled packet or the first packet. Information about how data is fragmented is contained within the IP packet header.

1.5 IP protocols and ports

> *protocol*
> 6c. In extended use: the accepted or established code of behaviour in any group, organization, or situation; an instance of this.
> 6d. Computing and Telecommunications. A (usually standardized) set of rules governing the exchange of data between given devices, or the transmission of data via a given communications channel.
>
> *"protocol, n.". OED Online. July 2018. Oxford University Press. http://www.oed.com/view/Entry/153243?rskey=WKMDkV&result=1 (accessed October 29, 2018)*

If IP addresses are analogous to the sender and recipient addresses on postal mail, IP protocols are analogous to the packaging that contains what's being transported. Protocols describe how IP protocol *payload data* is encapsulated, and sometimes includes additional data hosts use to determine the recipient. While the general idea behind each protocol is the same (move data from point A to point B), they each have distinct characteristics. The most commonly used protocols on the Internet are TCP and UDP.

TCP AND UDP TRAFFIC accounted for at least 90% of all traffic passing into and out of UMN, for a randomly sampled week in October 2018. This does not include TCP or UDP traffic encapsulated within another IP protocol.

Each protocol has a different structure, although they may share some common traits. Because TCP, UDP, and Internet Control Message Protocol (ICMP) account for the bulk of traffic traversing networks, we will focus on them. However, many of the principles discussed apply to other IP protocols (like GRE, Authentication Headers (AH), or Encapsulating Security Payload (ESP)).

Protocols like TCP and UDP use *ports* to match packets being transported with their intended recipients. The term originates from the physical world, where a port is an endpoint for interconnection between physical devices. For devices to communicate via physical ports, one device's port needs to be physically connected to the other.

TCP and UDP operate in a similar way, with numeric identifiers substituting for physical ports. A process that is assigned a port uses that port to send or receive communications with other processes. The IP address tells the network where to send traffic and the port tells the receiving computer which program is supposed to receive the payload being transported. Both protocols can technically use ports from 0-65535, but it's rare to see legitimate traffic originating from or going to port 0.

There are a couple ways ports are categorized. Traditionally, ports under 1024 are considered "privileged" ports, as special privileges are needed to *bind*[19] to these ports. All the other ports (>= 1024) are unprivileged ports, usable without special privileges. Although the distinction has

[19] "Binding" is the term used to describe opening a port to send or receive data.

faded over time, most modern OSes still differentiate between privileged and unprivileged ports to some degree.

Another common way ports are categorized is by whether they're "well known." Well-known port numbers are assigned by the Internet Assigned Numbers Authority (IANA) for specific network services. HTTP, SMTP, and DNS all have well-known port numbers. This simplifies the task of connecting to the correct port, as everyone knows (by convention and IANA's publication) which ports are associated with which services. You could run a service on a different port, but doing so may make it much more difficult for legitimate users to locate and use a service if you don't inform them of your nonstandard configuration. You can also run a service on a well known port assigned by IANA to run a diffferent service, but this could cause confusion. For example, mail delivery via SMTP is unlikely to work if the service listening on the port assigned for SMTP is running a web server instead.

A third way ports are categorized is by whether they're "ephemeral." Ephemeral ports are selected, used, and discarded as needed. Depending on conditions a computer may reuse the same ephemeral port repeatedly for different connections, but it's more likely to choose a new one for every connection. Because they're randomly selected, ephemeral ports make no sense for providing a service to the Internet, but are routinely used as source ports for programs or users attempting to connect to a service listening on another port. For example, when your web browser connects to a website, it is assigned an ephemeral port by the OS and opens a connection from that port to the port where the web server is listening. Communications are passed by IP, but each host uses the destination port in the IP protocol header to determine which program is supposed to receive the data being transmitted. The source and destination ports are always relative to the way the network traffic is moving. Traffic from an ephemeral port used by your web browser goes to port 80 on the web server. Return traffic originates from port 80 on the web server and goes to the ephemeral port originally used.

USING NONSTANDARD PORTS DOES NOT PROTECT YOUR SERVICES from being accessed by someone else. It may make it slightly more difficult to discover, but UMN is constantly being scanned for services listening on any port. Identifying the type of service takes a little work, but there are automated tools to do this. For example, running a web server on port 8080 instead of port 80 (the standard HTTP port) only slightly raises the level of difficulty someone attempting to find the web server faces. If you want to prevent access to a service, don't rely on obscurity for protection. Use a firewall.

1.5.1 TCP vs. UDP (and just about everything else)

TCP and UDP share the concept of ports, but that's about where the similarity ends. Unlike most other IP protocols, TCP provides reliable delivery of data, guarantees that it will be provided in the order in which it was originally sent, and can automatically request data that is lost or corrupted in

transit. A TCP connection appears to be a single, full-duplex connection to the application using it, but is actually comprised of two distinct, related *flows*.[20]

TCP sessions are started in a very specific way. The originator of a TCP connection sends a randomly selected number called a sequence number in a packet with a special SYNchronize (SYN) bit, or flag, set. The responding side sends a packet back, also with SYN set, an ACKnowledge (ACK) of the originator's sequence number, and its own randomly selected sequence number. The initiating side then ACKs the sequence number picked by the responding side and the session setup is complete. This is called a three-way handshake, often known by the TCP flags "SYN, SYN+ACK, ACK". Every subsequent bit transferred is ACKed by the side receiving it.

TCP also does special handling for shutting down a connection. Either side can close its half of the TCP connection, but cannot close the half opened by the other side. Doing so violates the protocol, which TCP indicates as an error. When one side closes its connection, it does so by sending a packet with the FINish (FIN) flag set. It must continue to receive data from the other side until the other side closes its half of the connection (also using the FIN flag). As with everything else in TCP, even the FIN packets are acknowledged.

TCP's design means it is best suited for applications that require guaranteed delivery or clear indication if delivery failed. E-mail, web traffic, and file transfers all benefit from TCP's features, because the software providing these services does not need to take into account whether packets arrive in order or at all. TCP ensures the data arrives and is in the correct order, and requests retransmission of parts that failed to arrive. However, these features come at a price. TCP requires more resources to use and isn't as fast at raw transport of data as some other protocols.

UDP also transports data between ports, but with none of the guarantees TCP provides. UDP has rudimentary data corruption detection, and does not provide guarantee, confirmation, or correct order of delivery. It is entirely up to the application receiving the data to account for this. This makes UDP much faster than TCP for raw data transmission at the cost of moving most error handling up the stack to the application. UDP is extremely well suited to tasks where loss of a single packet have little effect (like in video streaming).

It's also suitable for applications in which an entire transaction fits into a single packet, e.g., with DNS. In DNS, name requests and replies can typically fit inside a single packet. If a request gets lost, the requestor will tire of waiting for a reply and resend the request automatically. If the response is too big to fit into a single packet, DNS has features which allow it to automatically switch to using TCP as needed. Network Time Protocol (NTP) is another application in which an entire request or response can fit into a single packet, and the low overhead of UDP is a boon to keeping accurate time.

[20] A *flow* on an IP network is defined as the transmission of data identifiable by the source and destination addresses, the IP protocol, ports (if applicable), the amount of data, and the time. Flows are always unidirectional, but pairs of flows may be correlated to identify bidirectional communications.

Chapter 2

Firewall types

> One person's "paranoia" is another person's "engineering redundancy."
>
> *Marcus J. Ranum*

firewall:
3a. Something designed to protect the security or integrity of a system, process, or institution, esp. by acting as a barrier; a measure taken to prevent something undesirable occurring; a safeguard.
3b. Computing. A system, typically a piece of hardware or software, which provides protection against unauthorized access to or from a private network or computer system.

"firewall, n.". OED Online. July 2018. Oxford University Press. http://www.oed.com/view/Entry/249400?rskey=zwGVda&result=1 (accessed October 30, 2018).

Firewalls are unlike most network equipment in that they are designed to deliberately block the flow of data through a network. Where network devices like routers generally look at just the destination address, firewalls can, and usually do, look at a lot more. Source and destination addresses, protocols and ports, payload content, or other aspects of the traffic attempting to pass may all be inspected, depending on their configuration and capabilities. The term "firewall" has changed over time to include a number of different methods for controlling traffic flow. The real-world postal counterpart is someone who inspects every package passing through their area of control and checks for irregularities in packaging, addressing, and/or content.

2.1 ACLs

ACLs are a feature available on many routers[1] for determining if a packet should or should not pass. Access control lists are stateless, i.e., they can be used to make decisions based on addresses, protocols, and ports of a packet but not based on other packets. These are some of the simplest forms of network controls that could be called firewalls, and date back to the late 1980s. They're sometimes called *first-generation* firewalls.

UMN USES ACLs IN SOME PLACES to block very specific traffic involving services that are known to be extremely vulnerable to abuse or misuse, or are not intended for use on the open Internet. These can be viewed online at `https://it.umn.edu/university-network-border-blocks`.

ACLs consist of the desired action (allow or prohibit) and one or more of:

- source address or address block
- destination address or address block
- IP protocol
- source and/or destination port(s)

Even if it might be technically possible on some devices, ACLs have zero practical use for controlling traffic flow using other criteria (e.g., packet payload). They should be considered mainly for controls based on network layers 3 and/or 4. ACLs are simple, but that simplicity usually requires a strong understanding of the protocols and applications you're attempting to protect. Their simplicity also makes it difficult to come up with truly robust policies. ACLs work on flows (see section 1.5.1 for more on flows), so a pair of complementary ACLs may be needed to properly allow bidirectional communications.

For example, for HTTP connections, two ACL entries are needed: one to allow the client to talk to a server, and another to let the server respond. An ACL policy permitting packets in only one direction is insufficient because ACLs require return traffic also have policies covering it explicitly defined. Table 2.1 shows an example of an ACL to pass HTTP traffic.

Action	**Source address**	**Source port(s)**	**Destination address**	**Destination port**
allow	`192.0.2.1`	`1024-65535`	`192.0.2.101`	`80/tcp`
allow	`192.0.2.101`	`80`	`192.0.2.1`	`1024-65535/tcp`
deny	any	any	any	any

Table 2.1: Example of a router ACL

The example ACL allows the client, `192.0.2.1`, to talk to the server, `192.0.2.101`. For this policy to work, the client has to use a source port in the range 1024-65535 and the server has to

[1]The term "router" designates a network device used in a commercial or industrial network. While the term is often used to refer to "gateway" devices in a residential network, the term here is not intended to refer to those.

listen on port 80. Because ACLs are stateless, the first line needs a complementary entry to allow return traffic. Return traffic comes from the server on port 80 and goes to the source port used by the client. The third line denies all other traffic attempting to pass. Source ports are restricted to those shown because normal clients are expected to use unprivileged ports for ephemeral ports.

Where this ACL fails is that it also allows `192.0.2.101` to initiate a connection to any port on `192.0.2.1`, as long it originates from port 80. It also fails to account for ICMP traffic that may be related to this connection, e.g., ICMP "host unreachable" or "time exceeded" messages. To account for these conditions, the ACL needs to be much more complex.

2.2 Statefulness

> *state*:
> 1a. The combination of circumstances or attributes belonging at a particular time to a person or thing; a particular manner or way of existing as defined by the presence of certain circumstances or attributes; a condition.
> 3e. Each of the possible distinct modes of existence of a physical system or device; the condition of a device that determines what output it produces for a given input.
>
> *"state, n.". OED Online. July 2018. Oxford University Press. http://www.oed.com/view/Entry/189241?rskey=9EtzWU&result=1 (accessed November 05, 2018).*

Stateful packet filtering, found in *second-generation firewalls*, is similar to stateless firewalls in that they are largely limited to just layers 3 and 4. These were first used in the late 1980s or early 1990s. Stateful packet filtering is similar to stateless filtering (ACLs), except the firewall tracks some elements of the *state* of a particular communication. TCP has clearly defined states (e.g., the states of a handshake or its synchronized state), and stateful packet filters track these states.

Action	Source address	Source port(s)	Destination address	Destination port
allow	192.0.2.1	1024-65535	192.0.2.101	80/tcp
deny	any	any	any	any

Table 2.2: Example of a stateful firewall rule

This allows for more refined and simpler filtering than a stateless firewall. Because they track state, stateful firewalls can automatically add implied rules to allow traffic related to traffic permitted by explicit rules. Building on the previous example, the policy needed to allow client `192.0.2.1` to connect to web server `192.0.2.101` would look like that shown in table 2.2. The first rule in this policy allows traffic from the client to the server, and the second blocks everything else.

Rules to permit return traffic aren't added because the stateful filter understands they're *implied* by the rule explicitly allowing the initial connection. When the client initiates a connection the firewall checks that the addresses, protocol, and ports are correct, but also checks to see if the SYN flag is set. It looks for a SYN on the first response and verifies the sequence numbers are valid, then passes the return traffic. It continues to monitor the TCP sequence numbers and the state of the connection (setting up, synchronized, closing, etc.). Once the connection closes or is aborted, the

firewall returns to blocking all traffic except that explictly allowed by policy. The server cannot initiate a connection through the firewall because policy doesn't permit it.

Other protocols present more of a challenge. In particular, UDP lacks TCP's clearly defined states, i.e., it lacks session synchronization, sequence numbers, session termination phases, and more. (While protocols running over UDP, like DNS, may have specific states or sequences of behavior, packet-filtering firewalls don't have the necessary awareness of those higher layers in the protocol stack and can't use them to determine state.)

However, many other protocols involve a lot of back-and-forth communication, e.g., when a client sends a UDP packet to a server, it's not unusual for the server to send back a UDP packet in response. Firewalls will often "fake state" for this traffic and add implied rules as they're needed. If a firewall rule allows host A to send a UDP packet to host B, the firewall will often allow reciprocal traffic consistent with the rule from host B for a short period of time. The window of time is dependent on implementation. While there is no hard standard for this across firewall vendors, 30 seconds[2] is not an uncommon window duration.

STATEFUL FIREWALLS are attached at each router core at UMN. These may be used by departments desiring firewall protection, and where firewalls are mandated by policy. See UMN's Device Firewall Standard (`https://policy.umn.edu/it/securedata-appj`) for policy information, and contact Network and Telecommunications Services (NTS) or Information Technology Systems and Services (ITSS) (Duluth only) for information about available firewall options.

2.3 Proxies

proxy:
3a. A person appointed or authorized to act on behalf of another; an attorney; a representative, an agent; a substitute.
5. Computing. A system that enables the indirect exchange of data between computers on a network; spec. = proxy server n. at Compounds 1b.

proxy server:
[*Computing*] a server that filters requests between a client application and a remote server, chiefly for the purposes of efficiency (in handling communication that it can process without forwarding to the remote server), or to restrict access to the remote server by providing a firewall.

"proxy, n.". OED Online. July 2018. Oxford University Press. http://www.oed.com/view/Entry/153573?rskey=NQcaKI&result=1 (accessed October 30, 2018)

[2]Packet Filter (PF) and IPTables use 30 seconds as a default value.

Proxies are another early type of firewall. They work exactly as their name implies: clients communicate with the proxy, which communicates with the server on the client's behalf. The client and server never directly communicate with each other. This can provide a very high degree of security, as some classes of attacks (e.g., those that exploit vulnerabilities in how operating systems process packets) can't traverse a proxy. Since clients and servers never talk directly to each other, proxies can also act as a hard point of separation between networks. By design, proxies can sometimes also enforce policy based on application-layer data.

Proxies are often used in concert with other controls, where the proxy is the only host permitted to communicate from within a protected network. They can be very effective when paired with ACLs that permit only the proxy's communications while blocking all other connectivity. Clients communicating to outside must use the proxy. The proxy can log all activity and, by virtue of proxying data, has an opportunity for complete application awareness. This also completely eliminates exposing the clients directly to external attack.

Most firewalls on UMN's network work by packet filtering, not proxying. However, one place proxies are used is in the Payment Card Industry (PCI) environment, where policy requires strict controls on web access. Packet-filtering firewalls cannot make decisions based on content about whether to pass traffic. A web proxy can, and permits access to explicitly approved sites (e.g., to obtain software or antivirus updates) while refusing to proxy unauthorized activity.

Proxies are not without their drawbacks. One of the biggest is they must be able to properly understand the protocol(s) being proxied. For example, a proxy written specifically for HTTP probably won't work for SMTP. Also, the act of proxying connections adds additional delay to overall data throughput.

2.4 Host vs. network firewalls

Network firewalls, or network-based firewalls, are devices that are placed in-path in the network with the intent of controlling packet flows passing through them. These are implemented in several ways, including using a commercial product designed specifically to control traffic flow (e.g., a Fortigate), using a commodity PC running software that provides filtering capabilities (e.g., a computer running FreeBSD with PF or Linux with IPTables), or by leveraging the capabilities of existing network equipment (e.g., using ACLs on a router). Network firewalls provide protection for usually more than one device and are independent of the devices they protect.

Host firewalls, or host-based firewalls, are software on a host that provides traffic control for that physical host. The firewall capabilities are usually part of the OS running on the host. Host firewalls provide protection only for the processes running on them, and do not protect other, distinct hosts. Firewalls on systems hosting virtual machines (VMs) might also protect those VMs, but this should not be assumed to be true.

Each type has its pros and cons. Perhaps the biggest advantages to host-based firewalls are low cost, ready availability, and ease of maintenance. They're usually bundled with the OS, so it's probably already installed and paid for. Software updates for it are included in OS updates. However, host-based firewalls are unique to each host, and maintaining host firewalls on large groups

of hosts may carry an additional management cost. Also, loss of control of the host implies loss of control of the host's firewall.

In comparison, network-based firewalls are almost always more expensive to purchase, require integration into the network somewhere (which means designing the network with the firewall in mind), and often involve ongoing costs (e.g., for subscriptions or maintenance contracts). Where they excel is in protecting multiple networked assets, as this is what they're designed to do. In addition to having a single point where firewall rules are managed, many in-network firewalls also provide at least some instrumentation to monitor network traffic passing through them. Perhaps the biggest advantage is that compromising an individual host can't directly affect the protection provided by an in-network firewall.

UMN POLICY REQUIRES FIREWALLS based on a number of factors, including data security classification and security level. See Data Security Classification (`https://policy.umn.edu/it/dataclassification`) in UMN's policy library for more information.

2.5 Next generation: application awareness

Proxies can have significant, sometimes complete, awareness of the details of traffic they control, including the full data payload, because they are specifically designed to understand it. They are largely incapable of handling traffic for which they're not designed, though, and introduce sometimes significant performance penalties or complexity to network designs. These factors contribute to them being less common than packet filtering firewalls. First- and second-generation packet filtering firewalls (packet filters for short) lack application awareness but function reasonably well for what they can do. While generally not payload-aware, they are relatively easy to understand, configure, and manage compared to proxies. Because they require little processing power compared to proxies, they also tend to be cheaper.

As computing power has increased over time, new capabilities have been introduced into *third-generation* (or "next-generation") firewalls allowing for "application awareness." These firewalls can unpack and inspect payload data to inform traffic control decisions, usually at line speed. This gives them an opportunity to be aware of application-layer data, like a proxy, but in a way that doesn't impact the rate of data transfer through the firewall.

Application awareness takes several forms. Some firewalls can identify malware as it traverses the firewall and block the flow upon detection. Some firewalls can verify that traffic on a specific port is actually the type of traffic that's expected on that port, e.g., they can prevent someone from running Secure Shell (SSH) services on nonstandard ports. Still others can monitor an HTTP connection for protocol violations, e.g., attempts to abuse the HTTP protocol in a way that might adversely affect a web server. In combination with additional in-network defenses, some firewalls can identify denial-of-service attacks through traffic and content analysis and automatically request reconfiguration of the network to mitigate it. For example, if a particular HTTP payload is known to be associated with a common denial-of-service malware, the firewall detecting it can request traffic associated with hosts sending the malware be diverted for additional inspection and/or mitigation.

Application awareness is based on the use of protocol decoders. These decoders interpret payload data in a way the firewall can use for making traffic control decisions. If an application doesn't have

a corresponding decoder on the firewall, it's unlikely the firewall will be able to make any but the simplest application-dependent decisions. Also, there are sometimes different interpretations of application standards, so it's possible a decoder written against one interpretation of a standard won't work as expected with applications developed against a differing interpretation.

UMN HAS APPLICATION-AWARE FIREWALLS protecting systems in its data center networks. Use of firewalls is not currently (January 2019) mandatory for systems located in the data center but is expected to be a requirement in the Systems and Device Maintenance Standard following the next policy update.
Application-aware filtering will be provided by default for all devices in the data center.

Host-based firewalls are increasingly including application awareness built in. Because the host has the ability to match specific network traffic with processes running locally, it is in the best position to make determinations about whether particular traffic should permitted to be sent or received by programs running on it. Windows and macOS both have the ability to restrict traffic by application, although neither really provides payload-aware filtering.

Some software packages, like `mod_security` for Apache or `libmodsecurity` in general, are sometimes called web application firewalls. However, these are unlike conventional firewalls, including regular host-based firewalls, because they provide security to just a specific service, i.e., `mod_security` and `libmodsecurity` protect web server software, not other services on the host running the web server. (A true host-based firewall protects the host as a whole, not just one service.) These application-specific firewalls are sometimes difficult to configure and tune precisely, but can provide a great deal of protection if time and effort are invested in correctly configuring them.

Chapter 3

Firewall policies

> There is no such thing as paranoia. Your worst fears can come true at any moment.
>
> *Hunter S. Thompson*

policy:
4. A principle or course of action adopted or proposed as desirable, advantageous, or expedient; esp. one formally advocated by a government, political party, etc. Also as a mass noun: method of acting on matters of principle, settled practice. (Now the usual sense.)

rule:
22. Computing. A conditional statement that controls the behaviour of a system in a particular situation.

"policy, n.1". OED Online. July 2018. Oxford University Press. http://www.oed.com/view/Entry/146842?rskey=yjVveB&result=1&isAdvanced=false (accessed November 05, 2018)

"rule, n.1". OED Online. July 2018. Oxford University Press. http://www.oed.com/view/Entry/168717?redirectedFrom=ruleset (accessed November 05, 2018)

The terms "rule" and "policy" can have several different meanings with respect to firewalls, depending on a number of factors (who you're talking with, specific context, etc.). For our purposes, "rule" refers to a specific statement that determines the firewall's behavior with respect to controlling traffic. "Policy" refers to the natural-language definitions of what network traffic is being controlled and why. It also refers to a collection of one or more rules that belong together. An example policy is, "Only SSH connections from UMN's hosts may pass through the firewall." The implementation of that policy would be a set of rules like:

1. `pass in quick proto tcp from <UMN_hosts> to any port 22`

2. `block in quick proto tcp from any to any port 22`

Considered together, the rules above are a policy, albeit one that shows the actual implementation and not the natural-language policy on which they're based.

Historically firewalls have gained a reputation for inexplicably disrupting legitimate network traffic. These problems can often be traced, at least in part, to poorly understood and poorly documented firewall policies. In fact, firewalls usually do exactly what they're configured to do, and many "problems" come down to a mismatch between what people think is in the implemented policy vs. what's actually there.

A firewall policy should first be defined in natural language. This helps inform decisions regarding what rules are actually implemented, as not all firewalls have the same capabilities with respect to implementation of policy. (See chapter 2 for descriptions of various firewall types and a discussion of their capabilities.) It also provides documentation to someone examining the firewall's configured policy during subsequent inspection. This may seem overly burdensome, but pays serious dividends when it comes time to troubleshoot connectivity problems, for a new employee to take over management of the firewall, or to respond to an audit.

Modern firewalls, like commercial ones from Fortinet or Palo Alto, include graphical interfaces that simplify implementation of a firewall policy. Additionally, many modern firewalls can make use of objects in policies. These objects are used in place of actual values (e.g., they represent the actual IP addresses, protocols, ports, etc.) in a given rule. This allows for more generalized policies that are often easier to read. Instead of adding an individual rule to account for each line in policy, similar rules can be collapsed into one based on objects containing all the necessary elements.

UMN uses Fortinet firewalls in its network. These have a graphical control interface that includes version control. You can see who made changes, when changes were made, compare differences in policies on the firewall, and more. Training specific to these firewalls is available by requesting it using the Voice and Data Service Request form in ServiceNow (`https://z.umn.edu/servicenow`).

3.1 What makes a sensible firewall policy?

A sensible firewall policy:

- should default to blocking traffic
- are driven by institutional policy
- can be directly linked to a natural-language policy
- is defined in simple, straightforward terms
- makes use of reusable objects
- includes descriptive comments

"Default deny" is the preferred default for firewalls. A "default allow" policy requires you to identify all situations where traffic should be explicitly denied, and it is generally impractical to attempt to enumerate all possible undesired activity in advance. A much easier task is to identify the services you intend to support, the traffic you expect to pass, and block everything else by default. This also makes auditing the firewall much simpler.

SOME GRANTS HAVE SECURITY REQUIREMENTS that include a firewall. Federal Information Processing Standards (FIPS) requires firewalls for some security levels, and funding can be denied if those requirements aren't met. A firewall with a "default allow" policy is more likely by design to fail a security assessment, something that might have financial costs associated with it.

Institutional needs and policies determine your firewall policy needs, e.g., which protections are required, recommended, or optional in your firewall policy. Constructing a firewall policy in natural language provides a framework that can be adapted to various technical implementations of that policy. The nontechnical implementation of your firewall policy should be understandable to anyone who may be called upon to maintain your firewall.

Making use of reusable objects in object-oriented configurations can save considerable effort in implementing and maintaining a firewall policy. If you have a set of servers that policy states require comparable protection by the firewall, representing those servers collectively with an object and using the object in a rule to implement that policy is easier than implementing one rule per server.

An example natural-language policy is:

- Our web servers provide HTTP and HTTPS to everyone.
- Our web servers allow administrative access via SSH from our admins' workstations.
- All servers must be able to send e-mail via a designated SMTP relay host.
- The Tomcat administrative ports (8000 or 8080 are both used) on webserver3 and webserver5 is accessible from our admins' workstations.
- UIS's scanners have to be able to access everything.

This policy is straightforward and should be understandable to everyone who maintains the firewalls. Several things can be represented together by objects, such as:

- HTTP and HTTPS can be grouped in an object named `svc_webservices`.
- Individual web servers can be included in an object named `hg_webservers`.
- Administrators' workstations can be grouped as `hg_adminworkstations`.
- Webservers running Tomcat that should be accessible from admins' workstations can be grouped as `hg_tomcatservers`.
- The Tomcat service ports can be grouped in an object named `svc_tomcat`.
- All servers can be added to group `hg_servers`, with web servers being added to the group by including `hg_webservers`.
- UIS's scanners can go in a group called `hg_uisscanners`.

Actual implementations would look more like:

- Permit any to connect to `hg_webservers:svc_webservices`.
- Permit `hg_adminworkstations` to connect to `hg_webservers:22/tcp`
- Permit `hg_servers` to connect to `mailrelay:25/tcp`.
- Permit `hg_adminworkstations` to connect to `hg_tomcatservers:svc_tomcat`.
- Permit `hg_uisscanners` to connect to any.

These host and service objects have names that help identify what they are (they start with `svc_` for service objects and `hg_` for host groups). If a new web server is added, it only needs to be added

to the appropriate host groups (`hg_webservers` and `hg_tomcatservers`, as appropriate). Objects don't make sense for services like SSH or SMTP because there's only one thing in the policy to group. Note that every line in the implementation maps directly to a line found in the natural-language version of the policy.

3.2 Vendor insanity and firewalls

Many vendors sell devices that are intended to be networked, but are not intended to be connected directly to the Internet. For these devices, a firewall is all but mandatory, even if the vendor's documentation doesn't explicitly state a firewall is needed. Devices like scanning tunneling electron microscopes or audio/video control panels are generally not intended to be directly accessible by the Internet at large, so their manufacturers don't expend much effort towards hardening them against the attacks such access brings.

Over 700 external hosts each attempted to connect to at least 1,000 distinct hosts inside UMN on January 20, 2019, a Sunday on a holiday weekend, but had their attempts rejected because no services were listening on their intended targets. This day was one of relatively low activity, as UMN was still on winter break. Even assuming each external host only probed 1,000 hosts only once (the actual number is much higher), that still comes out to more than 8 probes/sec. UMN's network is **constantly** probed for weaknesses from all over the globe.

There are a variety of reasons vendors might downplay the need for and importance of a firewall. Their own research and product development networks are generally not exposed to the Internet, and they might assume their customers' networks are like theirs. They don't (can't!) know or understand every one of their customers' networks, and may not understand how well or poorly protected those networks are. They also rarely have any sort of direct, vested interest in the security of their customers' networks, particularly after they've completed a sale.

When evaluating vendors' claims or reading product documentation, there are several things to keep in mind. First, most vendors who claim the presence of a firewall will unquestionably cause their products to not work probably sell products that should be overlooked in favor of products from vendors who actually understand networks. It may be true that a complex firewall policy is required, but a blanket declaration to the effect of "firewall bad" (especially if they lack a clear understanding of the network configuration where the product will be used) signals basic ignorance of modern networking best practices. Often claims that the use of any firewall will completely break all functionality are baseless and wrong.

Second, vendors who tell you a firewall is completely unnecessary probably do not understand your network, or research and education (R&E) networks in general. These networks are very frequently configured nothing like a typical corporate network. Most corporate networks, including corporate research networks, have strong security controls, i.e., firewalls and other technology, in place to protect them from malicious activity. Their networks may be accessible from the outside via virtual private network (VPN) or similar, but usually not directly. In contrast, many R&E networks lack these controls. Devices connected to R&E networks are often directly connected to the Internet,

which means the Internet is also directly connected to them. Some in-network control is needed to protect them.

As of January 2019, firewalls are still the exception and not the rule on UMN's network. ACLs provide some basic protections, but connectivity is generally uncontrolled into and out of the network.

Finally, vendors cannot understand all the possible firewall and network implementations of their customers, so may have bad or no guidance regarding how to protect their products from network-based attacks on your network. The safest, sanest approach is to determine who needs what access based on general policy, tailor your firewall policy to match that, then define rules that implement your firewall policy. In other words, your devices or its manufacturer should not drive your firewall policy. Policy and your access needs should.

UIS provides technical consulting services. If you would like assistance in evaluating the security needs of a device you plan to connect to UMN's network, contact `security@umn.edu`.

3.3 Is a firewall really necessary?

Historically firewalls could have a drastic, adverse impact on network performance, but this has become much less common in the past 10 or more years. Packet filtering firewalls, even inexpensive ones built on commercial, off-the-shelf (COTS) hardware, have been able to keep up with gigabit networks for all but the most extreme edge cases. Ten-gigabit networks pose more of a challenge, but COTS hardware is also sufficient for most applications at those speeds, too.

Situations where a firewall is completely inappropriate are quite rare, and mainly involve extremely high-performance, extremely low-latency network applications at or above 10 gigabit. Even for these situations, however, firewalls (possibly in the form of router ACLs) are often still used, because some degree of firewall protection is required (e.g., by policy, requirements that are part of a grant, or common sense).

Those with a legitimate need for extremely high-performance network connectivity should review their situation with architects from UIS or those who run your network (e.g., NTS or ITSS). These resources can also help you realistically determine if a firewall likely to cause adverse effects and what options for in-network protection make sense.

Firewalls are also not a panacea for all your security concerns. They are intended to perform tasks specifically related to controlling traffic flow based their configured policies. Firewalls can have

varying capabilities, so you need to ensure the protection you think a firewall is providing actually exists. Firewalls should be part of a comprehensive security plan for your environment, one of several tools you use to protect your computing and network resources and data.

Chapter 4

Troubleshooting

> Everything is being run by computers. Everything is reliant on these computers working. We have become very reliant on Internet, on basic things like electricity, obviously, on computers working. And this really is something which creates completely new problems for us. We must have some way of continuing to work even if computers fail.
>
> *Mikko Hypponen*

Firewalls are sometimes blamed for connectivity breakdowns, but the firewall is usually just doing what it was configured to do. The ultimate cause can often be traced to a lack of understanding of their operation, inaccurate implementation of policy, and other related reasons. That said, it is frequently helpful to eliminate the firewall as a possible cause of problems, and there are a number of ways to do this.

For situations where connectivity was previously functioning, determine what changed (if anything) on the firewall. If nothing on the firewall was changed, it's unlikely to be the direct cause of new connectivity problems. It is entirely possible that its configuration may need to be altered to accommodate changes made elsewhere, e.g., changes in network traffic that resulted from reconfiguring a network endpoint, but it's rare for firewall configurations to spontaneously change.

The next step is to look at the firewall's logs and configuration. Firewalls usually can log information about traffic they permit and deny, although variations in policy implementation may result in that logging being disabled. If you are uncertain whether logging is enabled, check your firewall's documentation and current policy. Note that firewalls will frequently not generate log entries for every packet that reaches them. They're most likely to log the start of a flow, and less likely to log the end of a flow. TCP flows are usually the easiest to spot, as they should have distinct characteristics marking their start and end (SYN and FIN packets, respectively). Since other protocols, e.g., UDP, lack state, firewalls may guess as to the beginning and end of a "session" and log accordingly.

If you're unable to determine if changes have occurred and don't have access to the firewall's logs and configuration, you can also troubleshoot using commonly available command-line tools. Common tools in Unix-like environments (including macOS) include `ping` and `traceroute`. Their equivalents on Windows are `PING.EXE` and `TRACERT.EXE`. These are usually easiest to use from a command line. Understanding the capabilities of these two tools is crucial to effectively using them.

4.1 `ping` and `PING.EXE`

The idea of "pinging" a host likely has its origins in active sonar. In active sonar, sound is emitted and the delay in receiving an echo of that sound is measured. (For classic, but not necessarily factually accurate, examples of how this works, see almost any movie that prominently features a submarine.) The network utility works in a similar way in that packets are sent and the delay in the response is measured. A key differentiating factor, however, is that replies to "ping" over a network are optional[1] and may depend on how the target or the network connecting the source and target choose to handle the packet. (Designers of submarines are likely working to figure out how to make sonar "replies" optional.)

Generally, and with no unusual alterations to the network or endpoints, pinging over the network is fairly straightforward. A host sends a probe in the form of an ICMP `ECHO REQUEST` to a target. Upon receipt of this request, the target sends an ICMP `ECHO REPLY`. Since the host sending the original packet can time the delay between transmission and reception, it can gauge the delay, or latency, of the network between it and its target. ICMP is stateless, i.e., there's no handshake, and the turnaround in receiving an echo request and generating a corresponding reply is usually only a tiny fraction of the total time. Multiple echo requests can be used to gauge packet loss over the network(s) connecting the endpoints. This is why `ping` often defaults to sending more than one probe.

There are several factors that can make pinging useless for troubleshooting purposes. As a security measure, several operating systems[2] can disable replies to echo requests from hosts not on their local subnet, or sometimes completely. These hosts might not respond to echo requests. Additionally, some networks may only allow explicitly enumerated sources of echo requests, blocking everything else by default. This is why `ping` is better able to demonstrate connectivity works vs. demonstrating it's broken. Successfully pinging shows the network path works in both directions (at least for `ping`), but unsuccessful tests only show that something may be wrong.

`-t`	ping forever until manually stopped
`-a`	attempt to resolve addresses to names
`-n #`	send "#" echo requests
`-i TTL`	manually set the IP packet's time to live (TTL)
`-4`	force use of IPv4
`-6`	force use of IPv6

Table 4.1: Common arguments to `PING.EXE`

[1] Technically, responses to an ICMP `ECHO REQUEST` might be mandatory, depending on your interpretation of the standards. However, devices can be configured to not support it, so practically speaking it's optional.

[2] Including Windows, macOS, Linux, and some variants of Unix

Using PING.EXE on a Windows host (tested on Windows 10) is straightforward. When run without arguments, the utility provides a list of options. Some of the more useful ones are shown in table 4.1.

-c #	send "#" echo requests
-n	do **not** resolve addresses to names
-t TTL	manually set the TTL
-4	force use of IPv4
-6	force use of IPv6

Table 4.2: Common arguments to ping

There are counterparts to these in the Unix-ish world, but the command arguments are different. Some commonly used arguments for ping on CentOS 7 are shown in table 4.2.

One notable difference is ping tends to resolve addresses to names by default, and provide an option to disable this feature. PING.EXE does the opposite, *not* resolving addresses to names by default. Another difference is ping tends to run forever by default instead of requiring an argument to do so.

Note that some Unix variants have ping and ping6 binaries for IPv4 and IPv6 testing, respectively. On those platforms ping may not have distinct arguments to force use of IPv4 or IPv6. See your platform's documentation for more detail. On Windows, PING.EXE handles both versions of IP.

PING.EXE and ping both have the ability to set a TTL. This is less important, but still sometimes useful, for use in pinging hosts. TTLs play a much greater part in troubleshooting when using traceroute.

4.2 traceroute and TRACERT.EXE

In IP networking, each packet has a TTL. The TTL is intended to keep packets from traversing the network forever, and serve as a way to identify packets that should no longer be passed along the network. IPv4's TTL field was originally meant to represent the number of seconds a packet had been in the network, i.e., an actual "time" to live. In practice, each router a packet passes through decrements the counter so it's not really time-based anymore. In IPv6 this field is more accurately referred to as the hop limit. TTL is used here to refer to this functionality in both IP versions.

As a packet arrives on a router, i.e., not the intended recipient but a device that's forwarding the packet on, the router decrements the TTL. If the result is zero, the router generates an ICMP TIME EXCEEDED message, sometimes including a portion of the original packet in the new packet's payload. The source address of this ICMP packet is the router's own address, and the destination is the source address on the now-deceased packet. Packets whose decremented TTLs are nonzero are forwarded normally.

traceroute leverages this behavior by sending packets with increasing TTLs, and usually sends three probes per TTL value. For example, the first three probes sent have a TTL of 1, the next three a TTL of 2, and so on. At each hop, the TTL is decremented and the packet is forwarded (if the TTL is still large enough), or it dies of old age and an ICMP TIME EXCEEDED message is sent back to the source.

`traceroute` uses high-numbered UDP ports for probes' destinations, usually starting at 33432 and incrementing by 1 for each probe packet sent. This method was chosen because the UDP ports in the target range (33432 through $[33432 + n - 1]$ for n probe packets) are not assigned to any well-known services and usually have nothing listening on them. When a packet arrives at a UDP port that has no listening service, the host receiving the packet sends back an ICMP `PORT UNREACHABLE` message with a destination of the original packet's source and a source of its own address. `traceroute` exits upon receiving `PORT UNREACHABLE` messages from the intended target.

Not all network equipment transporting packets change the TTL. In particular, switches moving traffic between virtual LANs (VLANs) do not modify the TTL because they're not actually routing traffic. Also, some devices configured to route "stealthily" (notably some Linux-based firewalls) may not change the TTL to avoid being detected inline. Similarly, targets of traceroutes that are configured to be "stealthy," i.e., not respond to incoming traffic, may break `traceroute`'s probing.

`TRACERT.EXE`, by default, works in a similar way, but uses ICMP `ECHO REQUEST` messages for its probes. Probes that expire en route elicit the same response as `traceroute`'s probes, but endpoints respond with ICMP `ECHO REPLY` instead of ICMP `PORT UNREACHABLE`.

4.3 Example troubleshooting scenarios

The following are several common (and maybe one or two less common) scenarios where connectivity is impaired or broken and ways to troubleshoot and isolate the cause. The examples are based on client/server relationships because that is the most common type of network connection, but many of the principles can be adapted to other types of connections. This is not intended to be an all-encompassing list of potential failure modes, but to familiarize you with some common ones and steps you can take to troubleshoot them.

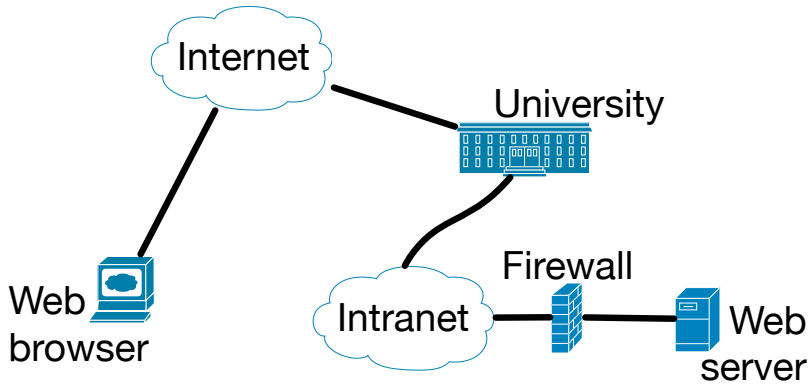

Figure 4.1: Basic network diagram used for included troubleshooting scenarios

The diagram in Figure 4.1 is intended for use with the troubleshooting examples, and illustrates what the network's known logical configuration is supposed to look like. It consists of a remote client somewhere outside the U connecting to the U via the Internet. Traffic reaching the U that's destined for the web server traverses the U's intranet, reaches the firewall, then (providing the firewall's policy allows it) passed to the web server. The implementation details of the Internet and the U's

intranet are unimportant in these examples, and are assumed to have no traffic controls in place that would interfere with routine HTTP traffic. (In practice, this may not always be the case.) In this diagram, the web browser is the client; the web server is the server.

4.3.1 Client fails to connect; failure is instantaneous

Symptom: The client cannot connect to the server. The connection is immediately or almost immediately refused.

HTTP runs over TCP, and TCP's connection-oriented protocol allows for an immediate rejection of a connection through the use of "reset" packets–TCP packets with the ReSeT (RST) flag set. An immediate response suggests RST is being returned in response to connection attempts. The Internet, U border, and U intranet aren't thought to have controls that would block a connection, so the logical points to check in the diagram above are the firewall and the web server. Normal TCP behavior requires a host to send RST in response to packets arriving on a port on which no service is listening. Therefore, things to check include:

- Is the service running on the server? Check to make sure the service is running (e.g., look at system logs and running processes).
- Is the firewall rejecting the traffic? Check the firewall logs for rejected connections.
- Is there a host firewall on the server that's rejecting traffic? Check the host's firewall configuration and logs.

For advanced testing, running `tcpdump` (or other packet capture software) on the firewall or the web server and attempting to initiate a connection can show if traffic is reaching the server and/or firewall. Note that because the connection is being refused immediately, this indicates the network is probably transporting traffic as expected.

4.3.2 Client fails to connect; delay in failure

Symptom: The client cannot connect to the server. There is a delay of more than several seconds before the connection fails.

TCP connection attempts that go unanswered will time out. Although the length of delay can vary by operating system, it is usually on the order of tens of seconds. Timeouts can be indicative of a firewall that's dropping (rather than actively rejecting with RST) TCP connections by policy, but could also be caused by any number of transient network failures anywhere in path between the client and server. Verifying basic connectivity is an important part of troubleshooting this problem.

The first, and easiest, tool to use is `ping`. On the client, attempt to ping the server. A successful response validates bidirectional connectivity, suggesting the problem may not lie in the network itself. It also shows the server is responding to at least some network traffic, meaning it's unlikely the server itself is down.[3]

It's sometimes useful to also do a traceroute, especially if the server doesn't respond to a ping. By default `traceroute` and `TRACERT.EXE` do not use TCP, but the network will treat their traffic the same as it does packets with a TCP payload. Output usually looks something like this (done from a Unix-like host):

```
% traceroute -n 134.84.23.118
traceroute to 134.84.23.118 (134.84.23.118), 64 hops max, 40 byte packets
```

[3]This may not be true in more complex environments, e.g., where a load balancer is in front of the web server.

```
 1  63.224.25.254  0.304 ms  0.233 ms  0.184 ms
 2  207.109.2.19  26.309 ms  22.976 ms  22.760 ms
 3  207.109.3.145  23.479 ms  22.993 ms  23.481 ms
 4  216.160.19.2  22.998 ms  23.490 ms  22.736 ms
 5  4.69.217.102  22.770 ms
    4.69.217.110  22.730 ms
    4.69.217.98  23.232 ms
 6  4.15.186.14  23.739 ms  24.492 ms  23.768 ms
 7  128.101.236.7  22.968 ms  23.975 ms  23.734 ms
 8  128.101.236.6  23.999 ms  23.224 ms  23.747 ms
 9  * * *
10  * * *
11  * * *
12  * * *
^C
```

Here's the output of `ping` from the client to the server:

```
% ping -c 5 134.84.23.118
PING 134.84.23.118 (134.84.23.118): 56 data bytes
64 bytes from 134.84.23.118: icmp_seq=0 ttl=53 time=24.069 ms
64 bytes from 134.84.23.118: icmp_seq=1 ttl=53 time=23.794 ms
64 bytes from 134.84.23.118: icmp_seq=2 ttl=53 time=24.117 ms
64 bytes from 134.84.23.118: icmp_seq=3 ttl=53 time=24.394 ms
64 bytes from 134.84.23.118: icmp_seq=4 ttl=53 time=23.415 ms

--- 134.84.23.118 ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 23.415/23.958/24.394/0.331 ms
```

Given that `ping` works, the network between the endpoints is known to work for at least some traffic. `traceroute`'s output (stopped manually with `<CTRL><C>` in the example) shows traffic getting to the U but not to the endpoint. This indicates the problem is probably within the U; in our diagram that would most likely be the firewall, the server, or possibly something in the U's intranet. The timing of packets seen by `ping` suggests the endpoint (or whatever might be responding on its behalf) is about 23-24 milliseconds (ms) away from the client. Since the last responding hop in `traceroute`'s output shows times slightly less than that, it's a reasonable guess that the firewall or server might be the problem.

Another thing to try is a traceroute using ICMP. Windows' TRACERT.EXE uses ICMP packets by default, but the Unix-like versions of 'traceroute' usually do not. The results using ICMP were more informative:

```
% traceroute -P icmp -n 134.84.23.118
traceroute to 134.84.23.118 (134.84.23.118), 64 hops max, 48 byte packets
 1  63.224.25.254  0.308 ms  0.211 ms  0.180 ms
 2  207.109.2.19  25.589 ms  22.461 ms  22.498 ms
 3  207.109.3.145  22.997 ms  22.982 ms  22.996 ms
```

```
 4  216.160.19.2  22.998 ms  23.480 ms  22.748 ms
 5  4.69.217.106  23.733 ms  22.782 ms  23.458 ms
 6  4.15.186.14  23.517 ms  24.966 ms  23.799 ms
 7  128.101.236.7  23.193 ms  23.739 ms  23.235 ms
 8  128.101.236.6  23.485 ms  23.499 ms  24.227 ms
 9  * * *
10  * * *
11  134.84.23.118  25.070 ms  23.688 ms  23.737 ms
```

The client can clearly reach the server using ICMP. Since the cause of connectivity failure seems to depend on the IP protocol in use (ICMP passes, but TCP and UDP appear not to), a logical culprit is a firewall, whether the one inline in the network or a host firewall on the server. In this case, the culprit is, in fact, the firewall, as its logs clearly show connectivity being denied.

One point to consider are the two "dead" hops in the traceroute, numbers 9 and 10. As you'll remember from section 4.2, the `ICMP TIME EXCEEDED` packets are created with the source address of the device generating them. Devices that have addresses that are not globally scoped (e.g., they're RFC1918 or similarly reserved addresses not intended for use over the Internet), the `TIME EXCEEDED` messages may be filtered out. Well-run organizations do not allow certain traffic to traverse their external border, and traffic sourced from addresses not used on the Internet are one type of traffic that should be blocked.

Another possible explanation is an in-network filter might be configured to prevent network devices from sending packets outside an organization. This is becoming more common in organizations with a strong need for security, as routers, switches, and other similar network infrastructure rarely need to communicate directly with outside hosts. Dead hops in a traceroute are not in themselves a bad thing, and it is sometimes worth the time to let a traceroute continue out at least past 20 hops if you are unsure of the network topology connecting the endpoints you're troubleshooting.

One last thing to try is `traceroute` using TCP packets. Some versions of `traceroute` do a better job of implementing this capability than others (e.g., FreeBSD's stock `traceroute` can send TCP packets but fails to look for any TCP-specific response).

4.3.3 Client can connect, but connections are very slow

Symptom: The client can usually connect to the server, but the act of connecting is very slow.

DNS is part of the lifeblood of the modern Internet. This is the mechanism by which names, which humans usually prefer, are resolved to numeric addresses, which computers require. DNS usually works in the background and is usually not considered by people during routine operation.

When DNS fails, however, many other things can also fail. In particular, anything requiring a connection using a name will likely be slow to complete when DNS service is degraded. Connections may also completely fail if DNS is responding too sluggishly and connection attempts are timing out. Situations where connectivity works reliably after the initial connection are often tied to DNS problems, i.e., the name lookup done at the time of initial connection will affect the speed with which the connection is established, but usually not anything after that. Note that clients that are establishing multiple connections, particularly web browsers, may have ongoing performance problems as they close old connections and open new ones.

An easy test is to try a host lookup from a command prompt. Windows systems include `NSLOOKUP.EXE`. That utility is also found on some Unix-like systems as `nslookup`, but it's been

deprecated in favor of newer utilities, like `dig` or `host` on many platforms. `NSLOOKUP.EXE`, `nslookup`, and `host` can all take a hostname or IP address as an argument. Output from `NSLOOKUP.EXE` and `host`, respectively, are:

```
>nslookup www.umn.edu
Server:  wbob-da-02.ggnet.umn.edu
Address:  192.168.251.2

Name:    www.umn.edu
Addresses:  134.84.119.107
          134.84.119.107


% host www.umn.edu
www.umn.edu has address 134.84.119.107
www.umn.edu has address 134.84.119.7
```

DNS responses should be close to instantaneous, and any delay of more than a few seconds may indicate a problem. This is especially true for repeated requests, as DNS includes a caching mechanism that should make subsequent lookups of a host or address faster than the initial one.

Failing DNS lookups are usually easy to spot.

```
>nslookup [DEAD DOMAIN]
Server:  wbob-da-02.ggnet.umn.edu
Address:  192.168.251.2

DNS request timed out.
    timeout was 2 seconds.
DNS request timed out.
    timeout was 2 seconds.
DNS request timed out.
    timeout was 2 seconds.
DNS request timed out.
    timeout was 2 seconds.
*** Request to wbob-da-02.ggnet.umn.edu timed-out


% host [DEAD DOMAIN]
;; connection timed out; no servers could be reached
```

Both of these indicate serious problems with DNS, and point to problems beyond the scope of the firewall or server to which the client is attempting to connect.

4.3.4 Client can connect, but connections die after a while

Symptom: The client can connect to the server, but the connection dies after a while.

Most TCP traffic consists of fairly short-lived sessions, usually lasting on the order of minutes to seconds. However, some TCP sessions are exceptionally long-lived, spanning days, weeks, or more.

Because modern firewalls track state for TCP connections, operations that cause the firewall to lose track of a connection's state (like rebooting the firewall) can cause connections to break. This sort of event is best identified by determining if maintenance is taking place on the firewall. However, some long-lived connections will appear to die even though the firewall is functioning normally, e.g., not undergoing work known to invalidate its state table.

This condition is often very hard to troubleshoot, as some firewalls may not log session termination. For example, this problem manifests itself in PF's logs like this:

```
14:13:16.517689 rule 22/0(match): pass in on red:
        192.0.2.1.16485 > 192.0.2.101.22: Flags [S],
                seq 3557621908, win 65535,
                options [mss 1452,nop,wscale 6,sackOK,TS val 875827367 ecr 0],
                length 0
14:23:03.224371 rule 0/0(match): block in on red:
        192.0.2.1.16485 > 192.0.2.101.22: Flags [P.],
                seq 3557626007:3557626059, ack 198150599, win 16402,
                options [nop,nop,TS val 876414072 ecr 1787470351],
                length 52
```

The first entry shows the initiation of a new session from the source (port 16485) to the destination's SSH port. The second entry, about 10 minutes later, shows a packet being blocked from that same source port. Besides the port pairings, the sequence numbers in the logs are helpful in correlating these two sessions. Note the initial sequence number when the connection was first passed: 3557621908. The sequence number in the packet that was blocked was very similar (3557626007 to 3557626059). Sequence numbers indicate the number of bytes transferred, so a low-volume connection that passes only about 4kBytes in 10 minutes ($3557626007 - 3557621908 = 4099$ bytes over 600 seconds, or slightly under 7 bytes/sec, or not much data) is a plausible explanation for what the logs show.

It may be easiest to diagnose this by looking at network traffic directly, e.g., with `tcpdump` or Wireshark. When a session times out, subsequent attempts by the client to send data will show as repeated transmission of the same packet or packets (identifiable as repeated transmission by the TCP sequence numbers). Repeated ACK packets of the same data are another good indicator.

A straightforward way to prevent idle sessions from being closed due to inactivity is to prevent them from being idle. Firewalls typically (but not always!) only look for activity at the TCP layer, i.e., if they see traffic for a session they assume it's not idle. The TCP protocol has a keepalive feature, and some software (including SSH) can make use of this to ensure TCP sessions persist. However, these can be forged, and often you might be more interested in ensuring the upper layer protocol is kept alive.

For protocols and services that support it, a keepalive function at the upper protocol layer is a better solution. In SSH, these controls are available in the client and server, and are `ServerAliveInterval` (client checks if the server's alive) and `ClientAliveInterval` (server checks if the client's alive), respectively. Both are often disabled by default.

Another is to enable any features your firewall might have for sending indications back that the connection is broken. Fortigates have an advanced rule option called `timeout-send-rst`, which causes the firewall to send a RST packet to client when their sessions time out. It's still up to the client to act on it, but this can provide some useful diagnostic data if you're troubleshooting a problem like this.

The default stance of your firewall may also have an effect. A firewall that returns an indication that traffic didn't pass through it effectively signals to the client exactly that. This is one of the more compelling arguments against "drop" (vs. "reject") for refusing traffic. Dropping unwanted or unexpected traffic destined for a client probably makes sense, as clients generally are not intentionally offering services for others' use. Actively rejecting unwanted traffic tells would-be attackers something's there, so silently dropping unwanted or unexpected traffic denies them that feedback. In contrast, servers by design are reachable over the network. They provide services. Actively rejecting unwanted traffic provides immediate feedback to anyone attempting to access the server that the attempt was unsuccessful. Sure, the Bad Guys[4] can use this information, but it has diagnostic value for legitimate users, too. For firewalls protecting servers, consider using "reject" for your default state. For clients, "drop" is probably a better default.

4.4 Other examples

There are a number of other not-uncommon connectivity problems you might encounter, but the troubleshooting examples covered in section 4.3 are useful in troubleshooting many of them. Here are a few other situations you may encounter.

4.4.1 Directional firewall rules

Most modern firewalls require the user to write rules that account only for the primary traffic intended to be permitted to pass. For example, if you want clients to be able to access a web server's standard HTTP and HTTPS ports, you would write a rule like:

```
pass protocol TCP from clients ports >1023 to server ports { http, https }
```

This rule takes into account the underlying protocol used by HTTP and HTTPS (TCP), that normal clients use ports selected in the unprivileged port range (ports >1023), and designates the standard TCP ports used for HTTP and HTTPS by specifying them by name. (For rules like this, firewalls will translate "http" and "https" to 80 and 443, respectively, as these are well-known service ports. You could also specify them numerically.) Notably absent are rules that allow:

- TCP traffic from the server to the client
- ICMP (to allow for "time exceeded," "fragmentation needed," or other messages)

Modern firewalls effectively add rules on the fly to account for possibly related traffic. Even with policies explicitly denying TCP or ICMP traffic from the server, traffic related to a session established under a rule like the one above will likely be allowed to pass.

However, traffic not related to an existing session, i.e., traffic the firewall can't associate with an entry in its state table, will likely not pass. For example, if the client connects from port 12345 to port 80, the server will not be able to send traffic from its port 80 to the client's port 54321. It will also be unable to send traffic from its port 443 to the client's port 12345, as that's not related to the initial connection to port 80.

Because TCP has clearly defined states, you are more likely to encounter problems with protocols like UDP, GRE, or other stateless protocols. Accounting for these may require manually implementing the rules that would normally be added reflexively, adjusting firewall timers regarding session tracking, or other adjustments.

[4] Whoever they are....

4.4.2 Transient connectivity problems

These are sometimes exceptionally difficult to troubleshoot. The main thing to keep in mind is that firewalls are generally very consistent in their responses. Transient connection problems may sometimes be traced to resource exhaustion on the firewall (see its logs for details!), but are more often cause by things like:

- a client's address changes and firewall policy doesn't account for it
- a client is choosing multiple paths for traffic (e.g., a laptop with wireless and wired connectivity)
- a wireless client is in a highly congested area
- resource exhaustion on the server
- network congestion or transient failure in-path

If other connectivity unrelated to the firewall is exhibiting transient problems, the problem is likely somewhere other than the firewall.

4.4.3 "Broken" client

Clients sometimes exhibit very broken behavior. While TCP and UDP behavior is fairly well defined (*de facto* if not in an actual standard), some clients may misbehave or behave in unexpected ways. Some network stacks do not have an awareness of the concept of "privileged" vs. "unprivileged" ports in TCP and UDP, and will select ports in the privileged range for ephemeral connections. Firewall rules like the one in subsection 4.4.1 will prohibit clients selecting privileged ports for ephemeral use from working.

Other problems you may encounter relate to less commonly used protocols, where implementations vary sufficiently that each may adhere to a "standard" in very different and incompatible ways. For example, the common task of reassembling fragmented IP packets varies between OpenBSD, Windows, Linux, and other platforms. This doesn't affect normal functionality of the protocols, as each platform is able to reassemble its traffic in ways that make sense to it. However, in cases where a firewall may be doing packet reassembly as part of payload inspection, its analysis may be based on a packet that's reassembled in an unexpected way.

A more common "broken" client is the one that simply isn't plugged in. Troubleshooting this situation is beyond the scope of this document.

4.4.4 Route loop

Route loops can manifest themselves as broken and transient connectivity. These are best diagnosed with tools like `traceroute` or `TRACERT.EXE`. Route loops are self-evident in the output of these tools: traffic will cycle between two or more devices. If one of the devices in the loop is the firewall, it could indicate a routing problem as opposed to a rule or policy problem. Enlist the aid of a networking expert to further troubleshoot these problems.

4.5 Other troubleshooting tools

`ping`, `traceroute`, and their Windows equivalents are far from being the only troubleshooting and analysis tools available. These are some additional tools that many people find useful. Check with your OS documentation or documentation for each tool to see if it's available for your platform(s).

`tcpdump`–A basic packet analyzer, `tcpdump` allows recording and analysis of network traffic sent from, arriving to, or passing through the host on which it's run. Traffic capture usually requires elevated privileges, but reading packet capture (PCAP) files can be done without special privileges. The PCAP file format originated with `tcpdump`, but many other tools are able to read and analyze it. Many people find it easiest to perform network captures with `tcpdump`, then move the resulting PCAP files to another host for analysis with other tools. A main advantage to `tcpdump` over many other tools is that it's bundled with the OS on some platforms (e.g., macOS) or very easy to install (e.g., via `yum` and `dpkg` on various Linux variants).

Wireshark–Over the past 15+ years it's been around, Wireshark's analysis capabilities have greatly improved to the point where it is the go-to program for detailed analysis of network data. While capable of directly capturing data on the host on which it's run, Wireshark is more often used to analyze PCAP files. In addition to doing basic IP packet and IP protocol decoding, it can analyze many higher layer protocols, strip payloads from captured data streams, and much more.

`mtr`–"Matt's traceroute," or `mtr` is a program that combines features of `ping` and `traceroute`, and does so quite successfully. It effectively combines the capabilities of these two programs into a unified display of network performance. Additionally, some versions are now aware of certain in-network features, and can identify things like multiprotocol label switching (MPLS) when it's in use.

Netcat and friends–Known by its short name `nc` on several platforms, Netcat is a tool designed to generate traffic and connect to other hosts via TCP and UDP. Flavors of Netcat have been enhanced on several platforms, but at its most basic it is an excellent tool for testing connectivity to remote systems. `nc` is standard on macOS. Tools like `socat` (common on several Linux variants), Cryptcat, Nmap, and others are decended at least in spirit from Netcat, and are also useful tools in their own right.

Chapter 5

Monitoring

> You can't defend. You can't prevent. The only thing you can do is detect and respond.
>
> *Bruce Schneier*

Monitoring your firewall's logs helps to reduce risk by giving you situational awareness of what your firewall is doing. This benefits you by allowing you to spot problems early, like identifying situations where the firewall may be blocking legitimate activity or not blocking activity that should be blocked. It also helps maintain an awareness of the current threats your online systems face, which may help inform future decisions you make as you add new devices or services to the network. Monitoring your firewall can help you reduce risk.

Firewall log monitoring falls under UMN's Log Management Standard (`https://policy.umn.edu/it/securedata-appm`), which requires logs be reviewed daily for anomalies for high security level data.

This chapter is incomplete.

UMN needs your help. There are many types of firewall, and many ways to monitor each of them. Some have graphical interfaces that can provide summaries of activities and allow drilldown to view the details of specific events, while others have no summarization capabilities and require command-line level access to view event details. Still others completely lack any easily accessible way to view event data or monitor their functionality. UIS needs *you* to identify the ones with which you want or need help, so training specific to them can be prioritized accordingly.

Appendices

Appendix A

Primer on binary, decimal, and hexadecimal

Humans use decimal (base 10). Computers usually work in binary (base 2), but it is sometimes easier to represent data in hexadecimal (base 16). This matches certain computer architectural details, and it looks close enough to base 10 for humans to use more easily. It's also a much more compact representation. Understanding their relationships makes it easier to work with the way IP addresses are sometimes represented and it's useful to know about bit positions for things like netmasks.

Hexadecimal numbers are usually written with a leading `0x` or `\x` to indicate they're hexadecimal. Binary numbers are usually explicitly identified as binary, but are sometimes identified by a following `b`, `B`, or subscripted `2`. We'll use `0x` and `b` here. Decimal numbers are usually just written normally, without additional identifiers. However, subscripting a `10` may be useful in situations where the base is not clear.

Numeric base	**Valid "digits"**	**Base 10 equivalent**
2	0, 1	0, 1
10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

Table A.1: Numeric bases and their relationships to base 10

Converting from another base to base 10 is fairly straightforward. Table A.1 shows the valid digits for the three commonly used numeric bases[1] You convert from binary or hexadecimal by multiplying each digit's value by the value its position represents, then adding the results. Figure A.1 shows this in detail.

Here's how you would convert the number `0x2A49` from hexadecimal to decimal. The base is 16 so, from left to right. The values are shown in table A.2. That's $8192 + 2560 + 64 + 9$, or `10825`. Binary conversion is similar.

[1] Octal, or base 8, used to be more common It's now mostly unused, so we don't cover it.

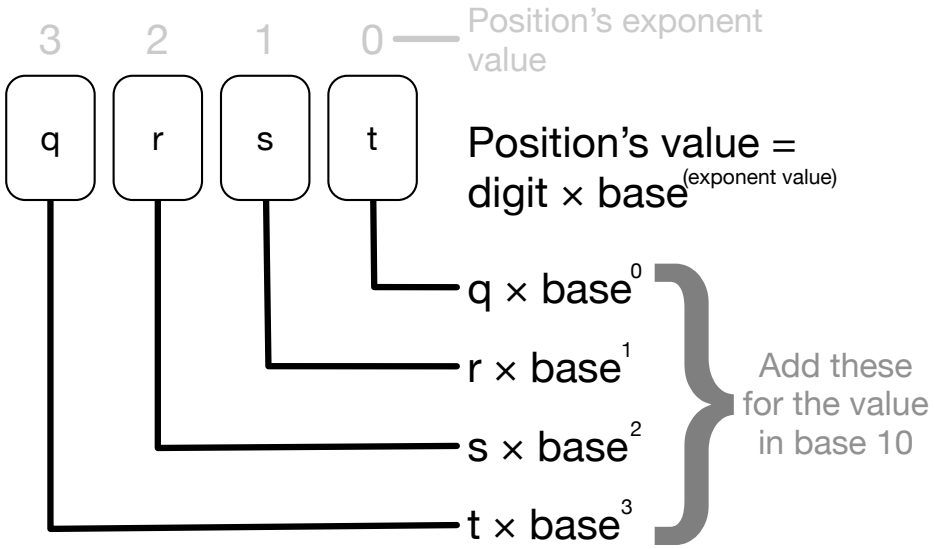

Figure A.1: Converting to base 10

Digit's shown value	Position's value in base 10
2	$2 * 16^3 = 2 * 4096 = 8192$
A	$10 * 16^2 = 10 * 256 = 2560$
4	$4 * 16^1 = 4 * 16 = 64$
9	$9 * 16^0 = 9 * 1 = 9$

Table A.2: Converting `0x2A49` to decimal

Converting `01101b` to base 10 is shown in table A.3.

That's $0 + 8 + 4 + 0 + 1$, or 13. Note the leading zero can be discarded, but sometimes leading zeroes are helpful to leave in place for formatting and alignment.

Going the other way is more complex. It involves finding the position with the highest value not exceeding the number you're converting, then subtracting the count of values for that position. You then repeat this for the next lowest-valued position. The best way to explain it is probably through example, so here's what the process looks like when converting the examples above back to their original bases, starting with converting `10825` to hexadecimal.

1. The base 10 values of hexadecimal number's digits, from lowest to highest are

   - 1s (16^0)
   - 16s (16^1)
   - 256s (16^2)
   - 4096s (16^3)
   - 65536s (16^4)

Digit's shown value	Position's value in base 10
0	$0 * 2^4 = 0 * 16 = 0$
1	$1 * 2^3 = 1 * 8 = 8$
1	$1 * 2^2 = 1 * 4 = 4$
0	$0 * 2^1 = 0 * 2 = 0$
1	$1 * 2^0 = 1 * 1 = 1$

Table A.3: Converting `01101b` to decimal

The fifth digit position is too large (> 10825), so we start with the fourth, the 4096's place.

2. The integer portion of 10825/4096 is 2, so the first digit (fourth position from the right) is `2`. Remove that from the value to look at the rest, i.e., $10825 - (2 * 4096)) = 2633$.

3. The integer portion of 2633/256 is 10, so the second digit (third place from the right) has a value of 10. In hexadecimal, `10` is represented by `A`, so that digit's value is `A`. Remove that from the value to look at the rest, i.e., $2633 - (10 * 256) = 73$.

4. The integer portion of 73/16 is 4, so the third digit is `4`. Remove that from the value to look at the rest, i.e., $74 - (4 * 16) = 9$.

5. The final digit (first place from the right) is `9`.

6. The result is `0x2A49`.

The process for binary is identical. Here's the shortened version:

1. The largest power of 2 that fits into 13 is 8, or 2^3. The first digit is in the fourth position and is `1`. The remainder is 5.

2. The next lowest power of 2 that fits into 5 is 4, or 2^2. This digit is a `1`, with a remainder of 1.

3. 2^1 is the next power to check, but it's larger than the remainer from the previous step. This digit's value is 0 ($0 * 2^1$), and the remainder is still 1.

4. The next lowest power of 2 that fits into 1 is 1, or 2^0. This means the final digit is `1`.

5. The result is `1101b`, which is identical to `01101b` (leading zeros don't affect value in any of these bases).

Appendix B

Walkthrough of a PF policy implementation

Seeing a firewall policy defined is probably the most illustrative way to show firewall policy development. Because of its text-based rule format, this example uses FreeBSD's PF. However, the principles used apply to any general firewall type.

Given: You have a server subnet containing a mail server and web server. The mail server needs to be able to exchange e-mail with the world at large. The web server needs to provide web pages to the world via HTTPS, but HTTP should only be accessible locally from the U because policy doesn't allow plaintext HTTP over the Internet. The mail server is a Unix host managed over SSH and the web server runs on a Windows server accessible via Remote Desktop Protocol (RDP). System administrators have an assigned subnet of their own from which they're authorized to connect to the servers, and any IP on that subnet should be able to reach the servers for administration. The mail server is already in place, but the web server is new and doesn't have an address assigned yet. The hosts also need to be able to use DNS and NTP. Future expansion plans include adding another, identical web server at some point.

A plain-language policy might look like:

- Allow the mail server to connect to anyone's SMTP port.
- Allow anyone to connect to the mail server's SMTP port.
- Allow anyone to connect to the web server's HTTPS port.
- Allow UMN to connect to the web server's HTTP port.
- Allow UMN's name servers to be accessed.
- Allow UMN's NTP servers to be accessed.
- Allow the subnet containing system administrators' computers to connect to SSH or RDP on the servers.
- Block everything else.

Things that could be defined using reusable objects include:

- server subnet
- workstation subnet
- networks that are used by UMN

- the SMTP service port
- the HTTP and HTTPS service ports
- the SSH and RDP service ports
- rules involving DNS and NTP
- the source ports connections should come from

The last item is often overlooked, but almost all modern operating systems choose ports in the range 1024-65535 for their source ports. Connections to services coming from ports below 1024 can be considered potentially hostile for most services.

This is an actual ruleset that implements this policy:

```
# Firewall policy for sample firewall
# Last updated 2018-11-05 by amesbury

# Define service ports and port ranges
unprivileged_ports="1024:65535"
p_smtp="25"
p_http="80"
p_https="443"
p_ssh="22"
p_rdp="3389"
p_dns="53"
p_ntp="123"

# Define some tables containing hosts/networks
# 192.0.2.0/24 - server subnet
# 198.51.100.0/24 - workstation subnet
table <WORKSTATIONS> { 198.51.100.0/24 }

table <SERVERS> { 192.0.2.0/24 }
table <MAIL_SERVERS> { 192.0.2.2 }
table <WEB_SERVERS> { }

# UMN's publicly routed IPs
table <UMN_pub_IPs> {
        128.101.0.0/16 131.212.0.0/16 134.84.0.0/16
        146.57.0.0/16 160.94.0.0/16
        2001:468:1900::/40
        2607:ea00::/32
}

table <UMN_priv_IPs> {  10.0.0.0/8
                        172.16.0.0/12
                        192.168.0.0/16
                        fc00::/7
                        fe80::/10
                }
```

```
# UMN services
table <UMN_timeservers> { 128.101.101.101 134.84.84.84 }
table <UMN_DNS> { 128.101.101.101 134.84.84.84 }


# Rules are *last* match!  Read as "X except Y."  For example, the
# first few rules are read as:
#       block all
#       ... except pass e-mail traffic from the mail server to anyone
#       ... except pass e-mail traffic from anyone to the mail server

block all

# Pass e-mail traffic for the mail server
pass proto tcp \
        from <MAIL_SERVERS> port $unprivileged_ports \
        to any port $p_smtp

pass proto tcp \
        from any port $unprivileged_ports \
        to <MAIL_SERVERS> port $p_smtp

# Pass HTTPS traffic from anywhere to the web servers
pass proto tcp \
        from any port $unprivileged_ports \
        to <WEB_SERVERS> port $p_https

# Pass HTTP traffic from anywhere at the U to the web servers
pass proto tcp \
        from { <UMN_pub_IPs> <UMN_priv_IPs> } port $unprivileged_ports \
        to <WEB_SERVERS> port $p_http

# Pass DNS requests from the servers to the nameservers
# DNS uses both TCP and UDP!
pass proto { tcp udp } \
        from <SERVERS> port $unprivileged_ports \
        to <UMN_DNS> port $p_dns

# Pass NTP traffic from the servers to the timeservers
# NTP can use unprivileged ports or the NTP port for source ports!
pass proto udp \
        from <SERVERS> port { $p_ntp $unprivileged_ports } \
        to <UMN_timeservers> port $p_ntp

# Pass administrative traffic from the workstations to the servers
```

```
pass proto tcp \
        from <WORKSTATIONS> port $unprivileged_ports \
        to <SERVERS> port { $p_ssh $p_rdp }
```

When parsed by the firewall, the comments are removed but the resulting firewall policy still includes enough information to be at least partially understandable (lines wrapped to fit the page):

```
unprivileged_ports = "1024:65535"
p_smtp = "25"
p_http = "80"
p_https = "443"
p_ssh = "22"
p_rdp = "3389"
p_dns = "53"
p_ntp = "123"
table <WORKSTATIONS> { 198.51.100.0/24 }
table <SERVERS> { 192.0.2.0/24 }
table <MAIL_SERVERS> { 192.0.2.2 }
table <WEB_SERVERS> { }
table <UMN_pub_IPs> { 128.101.0.0/16 131.212.0.0/16 134.84.0.0/16 146.57.0.0/16
        160.94.0.0/16 2001:468:1900::/40 2607:ea00::/32
        }
table <UMN_priv_IPs> { 10.0.0.0/8 172.16.0.0/12 192.168.0.0/16 fc00::/7 fe80::/10 }
table <UMN_timeservers> { 128.101.101.101 134.84.84.84 }
table <UMN_DNS> { 128.101.101.101 134.84.84.84 }
block drop all
pass proto tcp from <UMN_pub_IPs> port 1024:65535 to <WEB_SERVERS> port = http
        flags S/SA keep state
pass proto tcp from <UMN_priv_IPs> port 1024:65535 to <WEB_SERVERS> port = http
        flags S/SA keep state
pass proto tcp from any port 1024:65535 to <WEB_SERVERS> port = https
        flags S/SA keep state
pass proto tcp from <WORKSTATIONS> port 1024:65535 to <SERVERS> port = ssh
        flags S/SA keep state
pass proto tcp from <WORKSTATIONS> port 1024:65535 to <SERVERS> port = rdp
        flags S/SA keep state
pass proto tcp from <MAIL_SERVERS> port 1024:65535 to any port = smtp
        flags S/SA keep state
pass proto tcp from any port 1024:65535 to <MAIL_SERVERS> port = smtp
        flags S/SA keep state
pass proto tcp from <SERVERS> port 1024:65535 to <UMN_DNS> port = domain
        flags S/SA keep state
pass proto udp from <SERVERS> port 1024:65535 to <UMN_DNS> port = domain keep state
pass proto udp from <SERVERS> port 1024:65535 to <UMN_timeservers> port = ntp keep state
pass proto udp from <SERVERS> port = ntp to <UMN_timeservers> port = ntp keep state
```

There's a few things of note in the parsed output. First, the rules are not in the order in which they appear in the configuration file. PF will attempt to optimize rule order, but does so in a way

that doesn't affect the overall protection provided. Second, the rules include additional options not included in the original configuration file, notably the "flags S/SA" and "keep state" options. PF adds these for clarity in the output as it keeps state by default. Third, rules that contained groups of things in curly braces (like the rule for the NTP traffic) are expanded into more than one rule. PF expands some rule parameters in curly braces into multiple lines, but allows for shorthand in the configuration file. Fourth, port numbers are converted to port names where possible. For example, the rule for HTTPS lists "https" instead of 443.

Finally, even though we don't know the IP address for the web server, the rules relating to it are created and implemented anyway. PF simply interprets the empty object (the table that will eventually contain at least one web server IP address) as empty, which won't match any traffic that reaches the firewall. When we know the IP address, we can add it to the configuration file and reload it. Alternatively, an automated process can modify the object without reloading rules, e.g., for a situation in which web servers are automatically started or stopped. Either way, there is no interruption in operation when these objects are updated.

The concepts shown here can be found in many other firewall products. Fortigates, for example, make very heavy use of object definitions in much the same way PF does, but takes it further by allowing services to more easily be defined comprehensively instead of as a single protocol or port. Also, with centrally managed firewalls (again, Fortigates are an excellent example) common objects can be defined centrally and inherited by departments' own firewall policies. This allows the definition of rules based on objects managed by someone else. For example, if an object identifying all of UMN's networks is defined centrally, all policies that use it will be automatically updated if the U's list of assigned addresses changes as long as the centrally maintained object is updated. This sort of object sharing and inheritance frees firewall administrators from the task of having to track changes to objects that can be managed by someone else.

Initialisms, Acronyms, and Abbreviations

ACK ACKnowledge.

ACL Access Control List.

AH Authentication Headers.

ARP the address resolution protocol.

bps bits per second.

Bps bytes per second.

CIDR Classless Inter-Domain Routing.

COTS commercial, off-the-shelf.

DHCP Dynamic Host Configuration Protocol.

DNS Domain Name System.

DoD Department of Defense.

ESP Encapsulating Security Payload.

FIN FINish.

FIPS Federal Information Processing Standards.

GRE Generic Routing Encapsulation.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

IANA the Internet Assigned Numbers Authority.

ICMP Internet Control Message Protocol.

IP Internet Protocol.

IPv4 Internet Protocol version 4.

IPv6 Internet Protocol version 6.

ISP Internet service provider.

ITSS Information Technology Systems and Services.

LAN local area network.

MAC media access control.

MPLS multiprotocol label switching.

ms milliseconds.

NAT Network Address Translation.

NTP Network Time Protocol.

NTS Network and Telecommunications Services.

OS operating system.

PCAP packet capture.

PCI Payment Card Industry.

PF Packet Filter.

R&E research and education.

RDP Remote Desktop Protocol.

RFC Request for Comments.

RST ReSeT.

SMTP Simple Mail Transfer Protocol.

SSH Secure Shell.

SYN SYNchronize.

TCP Transmission Control Protocol.

TTL time to live.

UDP User Datagram Protocol.

UIS University Information Security.

ULA unique local address.

UMN the University of Minnesota.

VLAN virtual LAN.

VM virtual machine.

VPN virtual private network.

WAN wide area network.