Networking Highlights

This section contains a few important concepts about networking that will be useful to understand as we turn our attention towards internet and web app security.

The term *internet* refers to a network of networks. When you send something on the internet, it travels from your device, first passing through network you are on, through possibly a few other networks until finally it arrives at the network of your destination, where it will bounce around a bit before getting to the destination. At each step of the journey is located a *router*, a device whose job it is to forward what you sent to other routers, with each step hopefully getting closer to the destination.

The information you send is broken up into small pieces called *packets*, often up to around 1500 bytes in size. Each packet starts with *headers* which contain information to help get the packet to where it's going and to help the recipient understand what is being sent.

Conceptually, it is helpful to think of network functionality as broken into layers. In the most common model, the *OSI model*, there are seven layers. The most important ones for our purposes are the *application layer* (layer 7), *transport layer* (layer 4), *network layer* (layer 3), and *data link layer* (layer 2). Layer 7 contains the networked applications we use all the time, like HTTP, email, and SSH. Layer 4 contains the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). Layer 3 contains the Internet Protocol (IP). Layer 2 contains protocols like Ethernet and Wi-Fi.

Application layer protocols

The application layer contains the networked applications we actually use, like HTTP, email, and SSH. We will look in detail at the HTTP protocol a little later. Another important layer 7 protocol is SSH (secure shell), which is used to remotely log onto a computer or device in order to run commands on it, transfer files, or do various administrative tasks. SSH replaces an earlier technology, Telnet, which is used for the same purposes but which is unencrypted.

Port numbers

At layer 4, one of the most important concepts is that of *port numbers*. When information is delivered to the destination, the destination computer needs to know what program to deliver it to. That's what the port number is for. For instance, if you contact a web server, you might send either a plain HTTP request or an HTTPs request. Typically your web browser will set the destination port number in the transport layer header to 80 to indicate plain HTTP and 443 to indicate HTTPs. Different processes on the web server handle each type of traffic, and the port number determines which process gets the traffic. Your browser will also set a source port number on that HTTP request usually to a random open port on your device so that when the server's reply comes back, your device's networking software will know to send the response to your browser. This also helps the browser keep track of that request since it typically makes many requests.

Certain port numbers are usually reserved for specific services. Here is a list of a few of the more important ones to know.

22	SSH
53	DNS
80	HTTP
443	HTTPs

There are many more port numbers reserved for various services. These reserved port numbers are more suggestions than requirements. You can run a server that listens for SSH connections on port 80 (the HTTP port), though that would generally be confusing. Sometimes people will run services on different port numbers from the standard ones. For instance, SSH is often a target of attackers trying to get into system. Sometimes people will move SSH from port 22 to some random high-numbered port. This cuts down on the number of attackers trying to SSH into the system, though it doesn't eliminate it.

Port scanners A *port scanner* is a program that sends traffic to a variety of ports on a server and listens to see for which ones it gets back responses. This can give an attacker information about what is running on the server, giving them potential ways into the system. A simple port scan might try port 1, then port 2, then port 3, etc. Probably the most popular port scanner is a tool called *nmap*. It's a free download, and it is bundled with some Linux distributions. Here is a an example of a port scan I did on a server I control. The name and IP of the server have been redacted in the output.

```
> nmap ---.---.---.---
Starting Nmap 7.91 ( https://nmap.org ) at 2022-10-14 16:32 Eastern Daylight Time
Nmap scan report for ----------.------.-----.com (---.---.---.---)
Host is up (0.048s latency).
Not shown: 997 filtered ports
PORT    STATE  SERVICE
22/tcp  open   ssh
80/tcp  open   http
443/tcp closed https
```

Role of the transport layer

One of the most important roles of the transport layer is to provide reliability. The layers below are unreliable in the sense that it's possible that a packet that is sent might not arrive at its destination. For instance, a router that it passes through might be overwhelmed with traffic and drop the packet (a layer 3 problem) or someone might turn on a microwave which can interfere with Wi-Fi signals (a layer 1 or 2 problem). The *Transmission Control Protocol* (TCP) is designed to detect when packets have been lost and to arrange for them to be resent. Exactly how it does that is beyond the scope of this introduction. TCP also provides flow control features to prevent a sender from overwhelming a receiver with data and congestion control features to help protect the internet itself from getting bogged down with excessive traffic. Most HTTP traffic runs over TCP, though not the newest version, HTTP3.

Besides TCP, the other important protocol at the transport layer is the *Uniform Datagram Protocol* (UDP). UDP is basically what you use if you don't care about any of the reliability features of TCP. It does the bare minimum of what a transport layer protocol needs to do. A good way to remember this is to think of the U in UDP is as standing for "unreliable." The reliability of TCP comes with a cost. TCP is slower, and if a packet is lost, TCP can hold up everything until the missing packet is resent and arrives. UDP, on the other hand, is sometimes called "fire and forget." It's what you use if you care about speed and can handle an occasional lost packet. A key application of UDP is streaming video.

The network layer and IP addresses

Layer 3's main concern is routing, namely figuring out how things should travel through the internet to get to the destination. The main protocol at this layer is the Internet Protocol (IP). Every machine on the internet has a numerical address, called its IP address. This is sort of like how everyone on a phone network has to have a phone number in order to be able to reach them.

IP addresses come in two flavors: IPv4 and IPv6. IPv4 addresses are written in dotted decimal notation consisting of four blocks of digits from 0 to 255. Some examples include 127.0.0.1 and 192.168.1.2. There are about 4 billion possible IPv4 addresses, though because of the way they have been divvied up, there are considerably fewer that can actually be used. To address the lack of IPv4 addresses, IPv6 was developed. These addresses are 128 bits in size, which gives a huge number of potential addresses. IPv6 addresses are usually written in 8 groups of 4 hex digits. An example is fe80::d0c9:1638:64a0:39ae. The double colon indicates several blocks of 0000 that have been omitted to keep the notation shorter. IPv6 usage has been slowly growing, and IPv4 remains the dominant type.

Special IP addresses In network security, a few IPv4 address ranges are good to know. The ranges 10.0.0.0 – 10.255.255.255, 192.168.0.0 – 192.168.255.255, and 172.16.0.0 – 172.31.255.255 are private addresses. These ranges are used on local networks. Millions of different businesses and households simultaneously use these ranges for their own private networks. Those addresses only have meanings on those local networks, and packets with those addresses are never forwarded out of the network onto the main internet. The key thing to remember here is that whenever you see an address in one of these ranges, that means it is part of a local network and is not globally accessible.

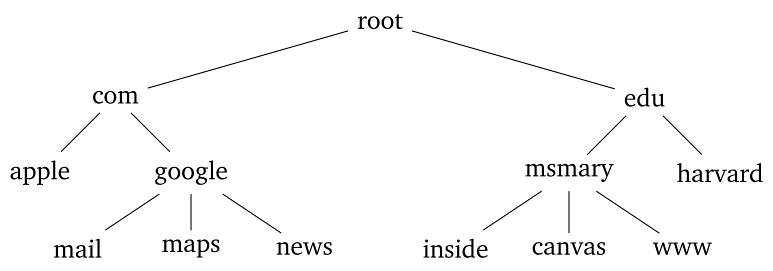
Another useful address range is 127.0.0.0 – 127.255.255.255. Any address in this range is called a *loopback address*. These are used for testing and running network applications on your own computer. These addresses always "loop back" to the computer itself and never go out to any network. The name *localhost* is often used to refer to a loopback address.

Security Whenever you visit a website, unless you take special precautions, your IP address, or at least the IP address of the network you are on (see later about NAT and VPNs), will show up in the website's logs. If you are managing a server and see certain IP addresses show up in your logs, there is no guarantee that the machine at that IP address actually made the request. IP addresses, along with just about everything else in packet headers, can be *spoofed*. That is, people can send out packets with a fake source IP address. When they do that, they won't be able to get a reply from your server since the reply will go to the faked source address, but sometimes attackers don't care about that. For instance, several types of *denial of service attacks* (DoS attacks) involve spoofing IP addresses. One simple kind of DoS attack floods a server with bogus TCP connection attempts, each with a spoofed source IP so that the source of the attacks is hidden.

DNS

DNS is the Domain Name System. IP addresses are not easy to remember, especially IPv6 addresses. They often change, and companies often maintain a bank of different addresses. To make things easier on humans, we use domain names. Domain names are things like `msmary.edu` or `maps.google.com`. DNS is the system that translates between these nice names and the IP addresses required by networking software to properly send packets around the internet.

This hierarchical system is laid out like a tree. At the top of the system is the root level. The first level below it contains what are called *top-level domains* or TLDs. Below them are individual organizations; each organization can split up their domain into further levels as needed. Below is a very small portion of the overall tree.



Note that the tree can go deeper than this, though most of the domains you'll run into are on the second or third level below the root.

Subdomains Each level is a subdomain of the next higher level. For instance, in `inside.msmary.edu`, `msmary` is a subdomain of `edu` and `inside` is a subdomain of `msmary`. As you go right in a domain name, things get more general. For instance, `inside.msmary.edu` is used for the part of the Mount's website that is for people on campus. On the other hand, `msmary.inside.edu` would be something on Inside University's domain (if there even is such a school).

This is important for avoiding phishing attacks. For example, if you're doing your banking with Wells Fargo, you might log on at login.wellsfargo.com. However, if you see a link in an email telling you to click on `wellsfargo.login.com` or `wellsfargo.bheinold.namespace.com`, then you should not do so. Notice the domain right next to `.com` is not `wellsfargo` in those cases.

Top-level domains The domains at the level below the root are called *top-level domains* or TLDs. For a long time, there were only a few of them (com, org, edu, net, gov, mil, and int). Now there are many more, like tv, pizza. Also, each country was given its own two-letter TLD. For instance, uk is the United Kingdom, ca is Canada, etc. Some of these TLDs are now used for other purposes, though the TLD is technically are still administered by that country. One example is the ly domain, run by Libya, but used for sites like `bit.ly` (a URL shortener) and `music.ly` (now known as TikTok).

DNS lookups Just about any time you visit a website, the first thing your browser will do is start a DNS query to find the IP address of the site. Let's look at how this happens. A program that performs a DNS lookup is called a *resolver*. Businesses and schools often have resolvers located on their campuses to do the lookups. ISPs have networks of resolvers. If you're at home on cable internet, your browser will contact one of the resolvers on your ISP's network.

DNS is a kind of distributed database with the information stored at various places around the world. No one machine knows all the DNS information. There is far too much and it changes too often. When a resolver does a query, it first contacts a root name server. These servers are in charge of knowing about where TLD servers are located. The root server will tell the resolver about the location of an appropriate TLD server. Then the resolver will contact that TLD name server. That server is responsible for knowing the locations of the name servers for the various organizations that use that TLD. For instance, the edu name servers know the locations of the name servers for various educational institutions. The TLD server will tell the resolver the location of a name server for the organization in charge of the site whose IP we need. The resolver will then contact that name server. Most of the time, that name server will return the IP address we are looking for, though some organizations have their name servers in a hierarchy and the process can go on for a couple of more steps.

This process involves a lot of back and forth, which would make things very slow. So once a resolver goes through this process, it will *cache* the result for a while, usually anywhere from a few seconds to a few days, depending on the site. Future requests for the same site will be served from the cache instead of going through the long process.

Two nice tools for doing DNS lookups yourself are `nslookup`, which is available on all major operating systems, and `dig`, which is available on Mac and Linux, though you can download a version for Windows.

Security There's a whole class of attacks called DNS cache poisoning that involve attackers intercepting DNS queries and returning bogus answers to them. It takes some skill to do these on an ordinary network, but it's relatively easy to if the attacker sets up their own Wi-Fi access point.

A typical way this happens is as follows: The attacker gets a USB Wi-Fi device for a few dollars that lets them use their laptop as a Wi-Fi access point. They also download a common DNS server software called Bind that lets them use their laptop as a DNS resolver. They set up shop somewhere in public and wait for people to connect to their Wi-Fi. When people connect, the attacker's access point tells the victim's computer that the resolver to use is the attacker's resolver. When the victim tries to go to something like Gmail, the attacker's DNS will instead give them the IP of a phishing site that looks just like Gmail. The attacker then will get the victim's login credentials.

Domain name security Remember that domain names get more general as you move to the right. So in `email.msmary.edu`, `email` is a subdomain of `msmary.edu`. However, in `msmary.email.edu`, `msmary` is a subdomain of `email.edu`. So even though the domain name might look like it belongs to the Mount, it doesn't; it belongs to `email.edu` (if there even is such a place).

Sometimes people buy up domain names that are small variations on a real domain name, like `gogle.com` or `googgle.com`, trying to catch people that misspell a domain name. Companies like Google probably own the most common variations on their name, but smaller companies often don't. Also, there are many TLDs now, so while I may own `brianheinold.net`, I don't own `brianheinold.com` or `brianheinold.io`.

Domain names expire, often every year, and sometimes companies forget to renew their domain names. If someone jumps in right after a domain name expires, they can grab it.

Another thing to be careful of is that domain names, at least when you put them into a browser's URL bar, can have special characters. For instance, the character *a* has character code 97. However, the Cyrillic lowercase *a* looks identical to the ordinary *a* but has character code 1072. In some browsers, if you visit a link that has an ordinary *a* replaced with a Cyrillic one, the domain name will not be the one you expect.

The hosts file Before contacting a resolver, your computer will look at a particular file on your computer called the *hosts* file. This file has a table of domain names and the IP addresses they map to. Here you can manually say which IP address a domain name on that computer should resolve to, even if it's not actually the real IP address associated with that file. For instance, you could look up the IP address of `hood.edu` and put an entry in your hosts file pointing `msmary.edu` to Hood's IP address. Then if someone tries to go to `msmary.edu` on that computer, they will instead end up at `hood.edu`.

A common use of this file is to "black-hole" certain domains, like tracking sites, by setting the IP to 0.0.0.0. This IP address goes nowhere, which has the effect of preventing that site from loading in your browser. Malware can also modify the hosts file. Sometimes they will black-hole domains associated with antivirus software. They could also put in entries for sites like gmail or your bank to point to the IPs of phishing sites designed to look like the real sites.

The hosts file is at `/etc/hosts` on Mac and Linux and at `windows/system32/drivers/etc/hosts` on Windows. Note that `etc` is a hidden directory in Windows.

Open resolvers Often your resolver is run by your ISP. On the Mount's network, we have our own DNS resolvers. Recall that these do the DNS lookups for us. If the Mount wants to block people from reaching certain sites, they can do so by having their resolver return bogus IP addresses for queries for those sites. Your cable company ISP could theoretically do this, and ISPs in other countries use this technique to block people from getting to banned sites.

One way to get around this is to use an open DNS resolver. Several companies run these. The most well known is Google's 8.8.8.8. Cloudflare runs one at 1.1.1.1, and there are others. You can go into your OS's settings to change your DNS settings to use one of these resolvers if you don't trust your ISP.

DNS cache poisoning The basic idea of this attack is to replace the answer to a DNS query with a bogus answer. The goal is often to direct people to a site other than that which they are trying to get to. If an attacker is on a local network with their target, this is pretty easy to do. The attacker sniffs traffic on the network, watching for DNS queries. When they see one, they send a reply that looks like it's legitimate, but it contains the attacker's desired IP address in the reply. As long as the attacker's answer gets there before the real answer, the attacker's answer will be accepted.

It's possible for an attacker to do this from a remote network, but it's considerably harder. In order for the attacker's answer to be accepted, they must have a transaction ID and port number that matches the original query. That information is plainly visible when sniffing traffic on a local network, but it's not available to a remote attacker. There are over 60,000 possible transaction IDs and over 60,000 possible port numbers, so an attacker could make a lucky guess, but that's not likely. However, the attacker can send a bunch of replies at once, and by a careful application of the birthday problem, it is possible to get this down to a manageable number. However, the attack is difficult on modern systems. But some older DNS systems used predictable transaction IDs and port numbers, which made the attacker considerably easier.

There many places an attacker could insert a bogus answer during the DNS query process. Here are a few:

1. Send a bogus reply to someone making a DNS request (this was discussed above)
2. Send a bogus reply to a resolver. This reply will be cached, and it will affect anyone who uses the resolver as long as the reply is in the cache.
3. Hack a resolver to put a bogus entry on it. Just like above, this affects anyone who uses the resolver.
4. Hack a company's name servers. If someone hacked the name servers for `msmary.edu`, then most people in the world who try to go to `msmary.edu` will get a bogus answer (at least once the entries in their local cache expire).
5. Hack a domain registrar's TLD servers. Each TLD is administered by a particular organization. Chances are the big TLDs like com and edu are well secured, but some of the more obscure ones like pizza or gs might not be. If an attacker is able to put entries into the pizza TLD servers, then people going to `emmitsburg.pizza` could be redirected to a phishing site that would collect their payment details.
6. Hack people's routers and change their DNS settings. Your home Wi-Fi router has a web interface where you can change various settings on it, including what to use as the network's DNS resolver. These web interfaces are protected by a username and password, but many people don't change them from the default manufacturer's values. Those values are easy to look up online, and are often simple things like admin/admin or admin/password. This could be changed by someone with access to your network or even remotely if your router has an easily accessible public IP address.

Evil twin access points An access point is a device people use to connect to a Wi-Fi network. For about $50, you can buy a USB device to turn your computer into an access point. You could then observe the broadcasts of a legitimate access point, copy all the details, and make your access point look just like the real one. This is an evil twin access point.

When people connect to an access point, that access point sends them various network details, like an IP address on the network and the address of the Wi-Fi network's DNS resolver. The person running the evil twin can also use free DNS software called BIND to turn their computer into a DNS resolver. So when people connect to their evil twin access point, the attacker will send them the address of the resolver running on their computer. Since the attacker controls this, they can put whatever entries they want into it.

The last step of the process is for the attacker to start up their access point and start spoofing deauthentication frames to boot people off the real network. These are easy to create. The attacker makes sure that their network has a stronger signal than the real one, and since their network looks just like the real one, the deauthenticated people's computers will try to connect to the attacker's network.

This is a fairly easy attack to pull off. It's one of the reasons you have to be careful using public Wi-Fi in places like hotels, airports, and coffee shops.

Solutions There are a few solutions to some aspects of DNS attacks. One is DNSSEC, which adds some cryptographic authentication to the process. However, even though it has been around for a while, it has some issues and has not been widely adopted.

A recent solution is DNS over HTTPs. Instead of going through a local resolver, your web browser will send the DNS request over HTTPs to an open resolver, like 8.8.8.8 or 1.1.1.1, who will return the answer back over HTTPs. The whole query is encrypted, and it would be quite difficult for an attacker to insert a fake answer into the process. However, DNS over HTTPs makes it harder for some organizations to use DNS to block certain sites, which they may do for security purposes. Other people are concerned with giving more power to big companies like Google over basic internet services like DNS.

A few other important networking details

ICMP The *Internet Control Message Protocol* (ICMP) is used for sending error and status messages through the internet. It is used by the important ping and traceroute tools discussed below, as well as for other things.

Because attackers can use the information gained from ICMP messages to learn details about a network, some network administrators will block some or all types of ICMP traffic on their network.

Ping *Ping* is a simple network tool, usually used for testing if a site is up and reachable. At the command line, we can type `ping google.com` to see if we can reach Google. If the site is reachable, we will get a ping reply back. If we get no reply, it either means the site is unreachable or that the site we are trying to reach has been configured not to respond to pings, which is pretty common.

Traceroute *Traceroute* is a tool built into most operating systems. It is called `traceroute` on Linux and Mac and `tracert` on Windows. It is useful for seeing the path packets take from a source to a destination.

DHCP When you join a network, you usually need an IP address on that network. The *Dynamic Host Configuration Protocol* (DHCP) is the process by which you get one. When you first join the network, your computer broadcasts a DHCP message and a DHCP server on the network will reply offering you an IP address. It essentially leases you that address for a fixed amount of time, though before the lease expires, your computer will try to reapply to keep that address.

Network Address Translation *Network Address Translation* (NAT) is a technology that allows multiple devices on a local network to share the same global IP address. Usually the network's gateway router will do the network address translation. That router has both an internal network address and a global address that the local network shares. When a machine on the local network needs to access a site on the internet, it sends a request that passes through the router. The router will rewrite the IP address on that packet with its own global address. It may also change the port number on the packet. It records all this information in a table, particularly the port number. When a response comes back from the remote site, the router looks up the incoming packet's destination port in its table to know which machine on the internal network to forward the response to.

NAT makes it difficult for an external site to start a connection with someone inside the network. Entries in the NAT router's table are made when someone inside the network tries to contact someone outside it. When someone outside tries to contact somebody inside the network, that packet will be dropped by the NAT router because there won't be a table entry there for the packet, and the NAT router won't know whom to forward the packet to. This makes certain networking things more difficult, like running a web server at home or playing networked games. It also adds a certain degree of security since attackers can't easily send unsolicited packets to devices inside the network. It also makes it hard for an attacker to get a map of what the internal network looks like. NAT does not provide perfect security, however, as information can still get through if the attacker picks a port that does correspond to an entry in the table.

MAC addresses

Every machine connected to the internet has an IP address. In the OSI model, this is at layer 3, the network layer. Below that, at the data link layer (layer 2), we have a different address called a MAC address. MAC is short for Media Access Control. This address is needed in order for a device to send a packet directly to another device. IP addresses are used for routing things around the internet, while MAC addresses are what is actually used when it comes time for one device to send things to another directly.

MAC addresses are written in six groups of two hex digits, usually separated by dashes or colons. For instance, AB:CD:EF:12:34:56 is a typical MAC address. The first three groups are the manufacturer code. In order to make sure no two devices in the world are assigned the same MAC address, each manufacturer of network devices can apply for a block of these addresses. For instance, C0:C1:C0 belongs to Cisco. They can assign MAC addresses starting with these three blocks to each individual device they manufacture. There are sites online that you can use to look up the codes. Wireshark will also show them.

MAC addresses can easily be changed in software. Usually it's just a short command at the terminal. Devices also have recently started using MAC address randomization. This is in response to MAC addresses being used to track people. The way that works is someone could set up a Wi-Fi access point in public or in a business. Whenever someone walks by with their phone, when their phone sees the access point, it by default sends its information to the access point, which includes the phone's MAC address. Someone setting up multiple of these access points can use it to see where someone has been. It does require an additional step of figuring out who belongs to that MAC address, but that can be done in various ways.

Virtual Private Networks

A *virtual private network* (VPN) is a way to securely allow remote users to access a local network. For instance, in our Cyber lab, we have some servers that are only accessible on campus. They don't have global IP addresses. However, a VPN allows access to them from off campus.

In order for this to work, the network needs a machine that runs VPN software. Remote users who want to connect to the network will send their traffic to that VPN machine, and that machine will then send the traffic onto the network. Suppose a remote user wants to access a database on the private network that is at IP 192.168.4.47. It creates a packet with destination 192.168.4.47 and it puts that packet into another packet whose destination is the VPN machine. This is sent over an encrypted connection to the VPN machine, which pulls out the packet destined for the database and sends it through the local network. This is an example of tunneling, where we put one piece of network traffic inside another. The encrypted connection to the VPN machine can be done with a variety of technologies. One possibility is TLS. Another is IPsec, which is a technology that encrypts data at the network layer. A more recent technology is WireGuard.

With a VPN connection, it's like your computer is on the local network, even though you're remote. VPNs are used for this purpose and also for privacy on the internet, specifically to hide your IP. There are various VPN services that you can use for this. The VPN service's IP address is what the sites you're accessing will see, not your IP address. This keeps the sites from seeing your IP, though the VPN service will still see it, so you have to trust the VPN service. This can also be used to get around sites that restrict access based on IP address. For instance, if you want to access a site that requires you to be in Australia, you can use a VPN service in Australia.

Firewalls

In the real world, a firewall is a device meant to keep fire out. For instance, cars have firewalls to separate the hot engine from the rest of the car. In a network, a *firewall* is used to keep certain traffic from entering a network. It can also be used to prevent things from leaving the network. Firewalls can be programs running on a computer or they could be a physical device.

Firewalls often operate by *blacklists* and *whitelists*. A blacklist is a list of "bad" things to be stopped. Anything not on the list is allowed through. A whitelist is a list of "good" things to allow. Anything not on the list is blocked. On the one hand, whitelists are more secure, as it's often hard to think of all possible "bad" things to put on a blacklist, and things often change rapidly. On the other hand, sometimes whitelists are so restrictive that they make it hard for legitimate users to get their work done. If the whitelist is too restrictive, those users will look for ways around the whitelist, which might end up making things more insecure.

A simple type of firewall is a *packet filter* that looks at packets and make decisions based on information in the packet headers. This includes things like IP addresses, port numbers, TCP flags, and the type of traffic (TCP, UDP, ICMP, etc.). For instance, you could set a packet filter to reject all traffic except to ports 80 and 443 or to allow all outgoing traffic except ICMP. A nice packet filter for Linux is iptables.

Packet filters mostly work at layers 4 and below, inspecting info in packet headers. An application layer firewall works at layer 7, actually inspecting the data of a packet. Sometimes this is referred as *deep packet inspection*. An application layer firewall can do more sophisticated things than a packet filter, like blocking all outgoing FTP connections or blocking all packets that contain "stackoverflow.com". HTTPs can make an application layer firewall's job harder since the data is encrypted. Sometimes the firewall will man-in-the-middle (MITM) the connection. Instead of an encrypted connection directly with the remote website, a user will have an encrypted connection with the firewall and the firewall will have an encrypted connection with the website.

BGP

The Border Gateway Protocol (BGP) is a routing protocol that *autonomous systems* (ASes) use to talk to each other. An AS is an organization that has a network consisting of a block of real (not private) IP addresses. These are usually fairly large organizations like ISPs and phone companies. Smaller ASes usually pay larger ASes to carry their traffic. The biggest ASes usually *peer* with one another, where they agree to carry each other's traffic without charging the other. BGP is what is used to route traffic over the internet and from network to network. Because the ASes are often in competition with each other, and because money is involved, BGP is rather complex. But the basic idea of it is that BGP routers will advertise to other routers which IP addresses they are in charge of, and they tell others whether or not they have a route to get to a specific destination. The problem is that there is not much authentication to this. Other routers implicity trust the information they are given.

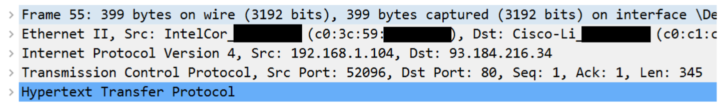
So if a BGP router claims that it has a certain block of IP addresses that it really doesn't, the other routers will start routing packets destined for those IPs to the router claiming ownership. This can make portions of the internet go offline for a while. It has happened multiple times where a country uses BGP to block access to certain sites within the country, and it ends up affecting other countries. This can also be used to "steal" IP addresses for a while, which can be used in DoS attacks or to view all the traffic destined for those addresses. People have used this with DNS cache poisoning to steal cryptocurrency. It takes some work to pull off such an attack, as not just anyone has access to these BGP routers. Generally, you would either have to locate one and hack it or convince an employee of an ISP to do your dirty work.

A few short demos

Here is an example of traceroute to see all the hops that a packet goes through from a source (my computer) to a destination (Google). This is `tracert google.com` at the Windows command line. Notice that the first address, 192.168.1.1 is in the reserved range for local addresses. That address belongs to my Wi-Fi router on my home network. The next address belongs to =Comcast. Traceroute tries to find domain names of routers as it finds them, and we see that starting at hop 6, some of the routers on the Comcast network do have names. Towards the end of the trace, we leave the Comcast network for another network and finally we end up at Google.

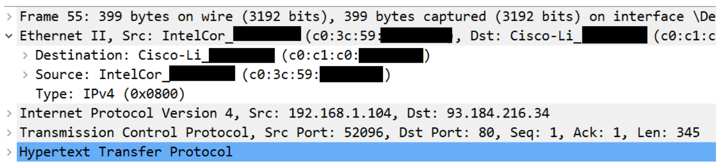
```
  1     1 ms     2 ms    <1 ms  192.168.1.1
  2    15 ms    13 ms    12 ms  96.120.9.9
  3    38 ms    16 ms    27 ms  162.151.69.213
  4    24 ms    24 ms    14 ms  96.108.5.201
  5    21 ms    22 ms    23 ms  be-34-ar01.mckeesport.pa.pitt.comcast.net [69.139.168.141]
  6    26 ms    23 ms    25 ms  be-31641-cs04.pittsburgh.pa.ibone.comcast.net [96.110.42.173]
  7    24 ms    22 ms    24 ms  be-1411-cr11.pittsburgh.pa.ibone.comcast.net [96.110.38.142]
  8    42 ms    28 ms    29 ms  be-302-cr12.ashburn.va.ibone.comcast.net [96.110.32.101]
  9    29 ms    28 ms    40 ms  be-1112-cs01.ashburn.va.ibone.comcast.net [96.110.32.201]
 10    28 ms    36 ms    28 ms  be-2111-pe11.ashburn.va.ibone.comcast.net [96.110.32.122]
 11    34 ms    30 ms    27 ms  50.248.119.106
 12    27 ms    27 ms    27 ms  108.170.229.246
 13    30 ms    31 ms    32 ms  209.85.251.83
 14    26 ms    26 ms    27 ms  iad23s63-in-f14.1e100.net [172.217.15.78]
```

A Wireshark trace Wireshark is a nice tool for watching network traffic on a device. Below is a capture of an HTTP message I sent to `http://www.example.com` The last four lines show the highlights of the headers from layers 2, 3, 4, and 7, respectively. Part of the MAC addresses have been blacked out.



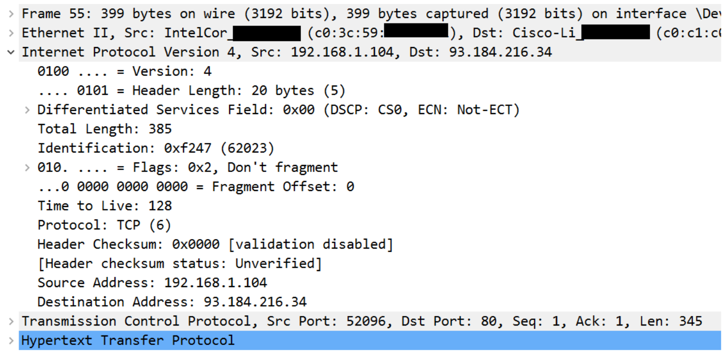
```
> Frame 55: 399 bytes on wire (3192 bits), 399 bytes captured (3192 bits) on interface \De
> Ethernet II, Src: IntelCor_[redacted] (c0:3c:59:[redacted]), Dst: Cisco-Li_[redacted] (c0:c1:c
> Internet Protocol Version 4, Src: 192.168.1.104, Dst: 93.184.216.34
> Transmission Control Protocol, Src Port: 52096, Dst Port: 80, Seq: 1, Ack: 1, Len: 345
> Hypertext Transfer Protocol
```

Below is the layer 2 header expanded. There's not much of interest in it other than the MAC addresses.



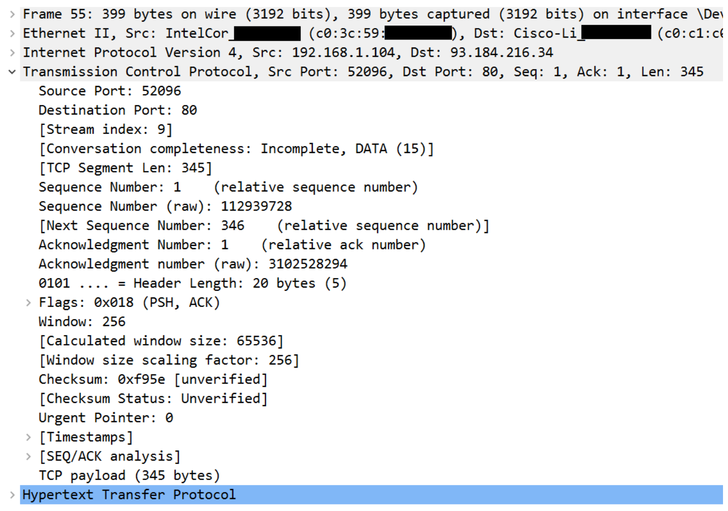
```
> Frame 55: 399 bytes on wire (3192 bits), 399 bytes captured (3192 bits) on interface \De
v Ethernet II, Src: IntelCor_[redacted] (c0:3c:59:[redacted]), Dst: Cisco-Li_[redacted] (c0:c1:c
  > Destination: Cisco-Li_[redacted] (c0:c1:c0:[redacted])
  > Source: IntelCor_[redacted] (c0:3c:59:[redacted])
    Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 192.168.1.104, Dst: 93.184.216.34
> Transmission Control Protocol, Src Port: 52096, Dst Port: 80, Seq: 1, Ack: 1, Len: 345
> Hypertext Transfer Protocol
```

Here is the expanded layer 3 header. There is quite a bit more here. Most of it is beyond the scope of this section on networking highlights, but notice the source and destination addresses at the bottom. The source addresses is in the 192.168 range, showing that it is a local address, specifically the address of my computer on my home network. When this packet gets to my NAT router, it will replace that local address with the router's own global address. The destination address is that of `example.com`.



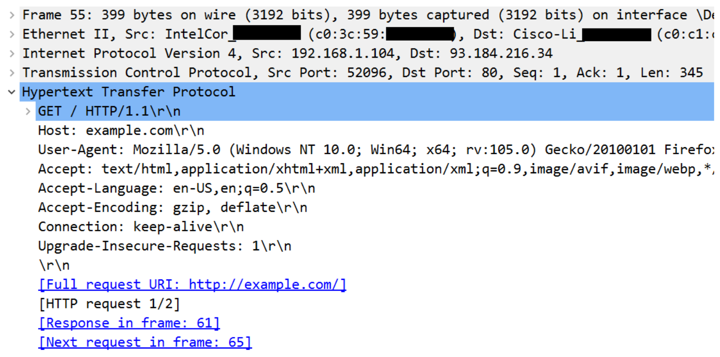
```
› Frame 55: 399 bytes on wire (3192 bits), 399 bytes captured (3192 bits) on interface \De
› Ethernet II, Src: IntelCor_[redacted] (c0:3c:59:[redacted]), Dst: Cisco-Li_[redacted] (c0:c1:c
⌄ Internet Protocol Version 4, Src: 192.168.1.104, Dst: 93.184.216.34
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  › Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 385
    Identification: 0xf247 (62023)
  › 010. .... = Flags: 0x2, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: TCP (6)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.1.104
    Destination Address: 93.184.216.34
› Transmission Control Protocol, Src Port: 52096, Dst Port: 80, Seq: 1, Ack: 1, Len: 345
› Hypertext Transfer Protocol
```

Next is the expanded TCP header. Just like the IP header, there is way more here than we can get into in this quick intro. But note the source and destination ports at the top. The source port, 52096, is a random open port my web browser chose. When a response comes back from the server, my networking software will use that port number to know to direct the response to my web browser, which in turn will know that it goes with my request to `example.com` and not some other request I might have made. The destination port is 80, which is the standard HTTP port, showing that this is an ordinary HTTP request.



```
› Frame 55: 399 bytes on wire (3192 bits), 399 bytes captured (3192 bits) on interface \De
› Ethernet II, Src: IntelCor_[redacted] (c0:3c:59:[redacted]), Dst: Cisco-Li_[redacted] (c0:c1:c
› Internet Protocol Version 4, Src: 192.168.1.104, Dst: 93.184.216.34
⌄ Transmission Control Protocol, Src Port: 52096, Dst Port: 80, Seq: 1, Ack: 1, Len: 345
    Source Port: 52096
    Destination Port: 80
    [Stream index: 9]
    [Conversation completeness: Incomplete, DATA (15)]
    [TCP Segment Len: 345]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 112939728
    [Next Sequence Number: 346    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 3102528294
    0101 .... = Header Length: 20 bytes (5)
  › Flags: 0x018 (PSH, ACK)
    Window: 256
    [Calculated window size: 65536]
    [Window size scaling factor: 256]
    Checksum: 0xf95e [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  › [Timestamps]
  › [SEQ/ACK analysis]
    TCP payload (345 bytes)
› Hypertext Transfer Protocol
```

Here is the expanded HTTP header. It contains info on what site we are looking for, as well as information about the browser making the request and information for the server about what the browser can understand.



```
› Frame 55: 399 bytes on wire (3192 bits), 399 bytes captured (3192 bits) on interface \D
› Ethernet II, Src: IntelCor_[redacted] (c0:3c:59:[redacted]), Dst: Cisco-Li_[redacted] (c0:c1:
› Internet Protocol Version 4, Src: 192.168.1.104, Dst: 93.184.216.34
› Transmission Control Protocol, Src Port: 52096, Dst Port: 80, Seq: 1, Ack: 1, Len: 345
⌄ Hypertext Transfer Protocol
  › GET / HTTP/1.1\r\n
    Host: example.com\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefo
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*,
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    \r\n
    [Full request URI: http://example.com/]
    [HTTP request 1/2]
    [Response in frame: 61]
    [Next request in frame: 65]
```