CSE127 Scribe Notes: Defending Networks

Spring 2021

These scribe notes were written by students in CSE 127 during Winter 2021. They have been lightly edited but may still contain errors.

1 Introduction

How do we harden a set of systems against external attack? It depends on the amount of network services your machines run: the more there are, the greater the risk. One approach could be to turn off the unnecessary network services on each system. But this can be difficult since we need to know everything about the services we are running, which is clearly not scalable to larger and more diverse network ecosystems.

The idea of "network perimeter defenses" tries to protect an organization's network from outside sources (i.e. the Internet). Such a system may include Firewalls, Network Address Translation (NAT), Application Proxies, and Network Intrusion Detection Systems (NIDS), among others. An assumption that this method makes is that the local network itself does not have any malicious actors.

2 Firewalls

With firewalls, we try to protect the network by isolating one part from everything else. Typically, we see this between our network and the global Internet. Sometimes, this can be used to quarantine infected machines within a single network.

A working firewall needs to filter or otherwise limit network traffic, but there are caveats: what information do you use to filter, and where do you actually perform the filtering? Different types of firewalls handle this differently. Two main types are the personal firewall, and the network firewall.

Personal firewalls run on end-hosts, like your own laptop. It has application and user-specific information.

Network firewalls, on the other hand, typically are in a machine between the outside (global Internet) and the protected network that routes the connections between the two. It tries to filter and protect against "bad" communications, and thus protects services inside the protected network from external actors

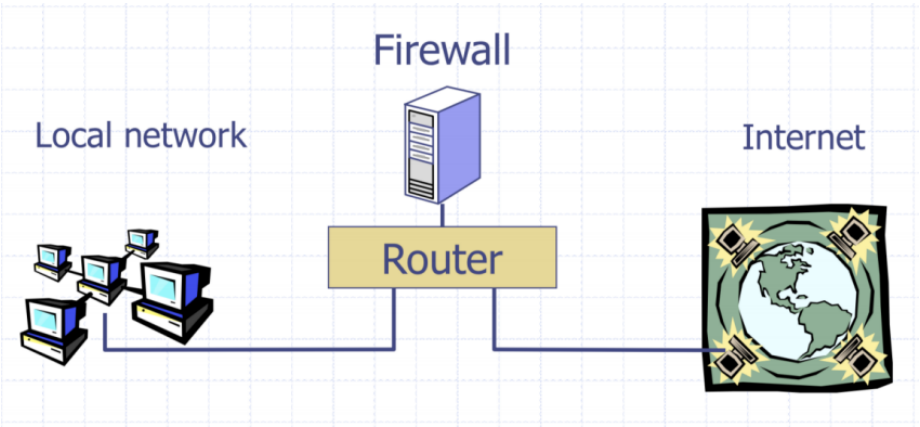

Figure 1: A good firewall will separate and protect the local network from malicious actors outside (such as from the Internet).

which may want malicious access. At the same time, it allows for outside services to be visible to hosts inside the protected network in a (hopefully) safe fashion.

2.0.1 Access Control Policy for Firewalls

One way firewalls can filter traffic is by enforcing an access control policy (ACP). Through it, the firewall monitors information on who is talking to whom and accessing what service. A very simplistic example allows all outbound connections while denying all inbound connections. This firewall would attempt to protect the network by allowing internal users to access anything outside, while denying external users access to anything in the protected network.

For the traffic that doesn't match rules, firewalls also need to be able to process network traffic which is not covered in the list of access control policies. The two generic policies are default allow (fail open) and default deny (fail closed), which by default allow and deny all unknown traffic received respectively. Of the two, default deny is much safer due to its conservative design. In case the conservative design hinders the working of the system, it would provide an early indication that an external service is important and the policy can be accommodated to allow the service.

2.0.2 Packet Filtering Firewalls

The simplest way a firewall can implement access control policy is by filtering the packets themselves. After defining a list of rules, the firewalls check every packet against the rules and decide to forward or drop it. These firewalls can take advantage of the information from network and transport layer headers, including Source and Destination IP, Source and Destination Port, and flags. This allows for better granularity in filtering network traffic. An example could

be to block incoming DNS (port 53), except for some known trusted servers. However, this type of stateless filtering also has limitations: it can't distinguish packets associated with a connection from those that aren't. Some firewalls have tried to solve this by keeping state about open TCP connections, which helps distinguish between external attempts at establishing communication with the local network and reply responses for communication established from internal users to the outside world.

2.0.3 Circumventing simple packet filtering

We can circumvent simple packet filtering in a couple different ways. A rudimentary idea could be to just send traffic on a port usually allocated for another service (e.g. running an SSH server on port 80 instead of the usual port 22). A more sophisticated method is through tunneling, which is to encapsulate one protocol inside another. The recipient of the outer protocol then decapsulates to recover the inner protocol. For example, we could encapsulate HTTPS requests inside of DNS queries, effectively allowing us to use HTTPS even if only the port for DNS is open. Note that for this example, we assume that the firewall is protocol-aware and filters non-DNS traffic over port 53.

2.0.4 Stateful packet filtering is hard

Let us take an example: suppose we want to allow inbound connections to a server, but block any attempts to log in as "root". How would we do this? What state do we need to keep?

An easy way to detect such requests would be to search for the word "root" in the incoming packet. Problems can emerge since "root" may be sent across multiple packets, and those packets could even be out of order. We can try to solve these simple problems by reconstructing the stream at the firewall.

But, even with a stateful TCP reconstruction, the attacker could bypass it using a Time to Live (TTL) evasion technique. An attacker can send multiple packets to the target machine such that several packets being sent have a TTL of somewhere between reaching the firewall and reaching the end host while the packets containing communication about getting root access could be sent with a much larger TTL to ensure that it reaches the end host. As a result, the firewall sees the entire message and forwards the traffic, but due to TTL expirations, only the malicious "root" sequence arrives at the internal host.

3 Network Address Translation (NAT)

The idea behind network address translation (NAT) is that IP addresses do not need to be globally unique, as long as they are within their own private networks. A NAT will map between local and global address spaces, so the outside will not be able to know what the actual IP is on the inside. Nowadays, most home routers are NATs and firewalls.

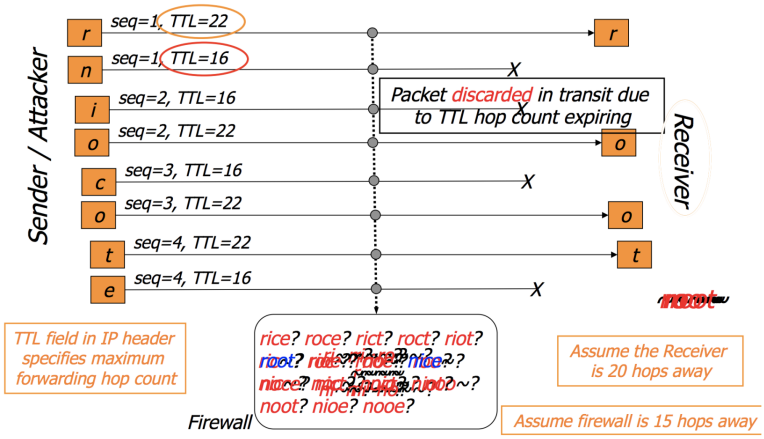

Figure 2: A diagram showing how packets will be discarded in transit (since the hop count has expired).

Typically a NAT will maintain a table with three main columns, namely, client IP, client port, NAT ID. This table stores a mapping from the local IP address and port number (local address space) to public facing IP address and a NAT ID (global space). Table entries expire after 2–3 minutes of no activity to allow them to be garbage collected. The table can be used to translate and process outbound and inbound traffic based on the following rules.

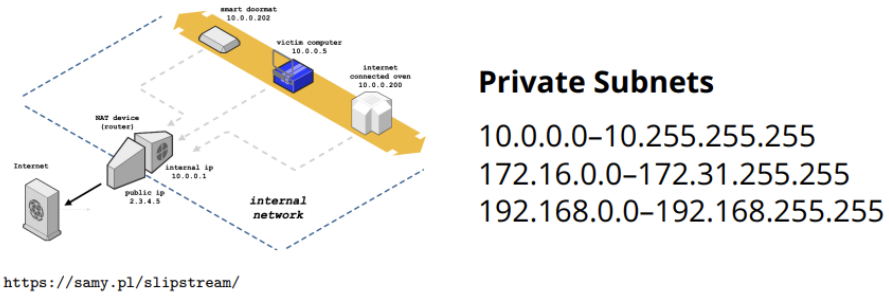

Figure 3: An example of a NAT'ed network. The outside internet does not know the actual IP used for any of the devices in the internal network.

Translating outgoing packets (on non-NAT port):

1. Look for client IP address, client port in mapping table
2. If found, replace client port with previously allocated NAT ID (same size

as port number)

3. If not found, allocate a new NAT ID and replace source port with NAT ID
4. Replace source IP address with NAT IP address

Translating incoming packets (on NAT port):

1. Look up destination port as NAT ID in port mapping table
2. If found, replace destination address and port with client entries from the mapping table
3. If not found, the packet should be rejected

The advantage of such a NAT table is that it only allows packets corresponding to connections from the outside that are established from inside. Hosts from the outside will only be able to contact internal hosts that appear on the table, and that entry is only created when a connection is established. Not only that, we don't need as large an external address space.

On the other hand, NATs come at a cost. Rewriting IP addresses is not so easy, as IPs may appear in the contents of certain packets in certain protocols. This mapping may also break certain protocols altogether, as the outside will not be able to initiate connections.

4 Application Proxies

Another idea to secure apps is by forcing them to pass through a proxy, which can be seen as an application level man-in-the-middle. These can enforce different policies for different protocols, such as scanning for virus and rejecting spam for SMTP and block forbidden URLS for HTTP. These can provide much more control in filtering network traffic as they access application level packet information.

An application proxy sits between the internal network and world outside. This is done to ensure that there is no direct communication between the internal network and the other end of the conversation. The proxy inspects the packets and decides which packets should be allowed to pass through.

To achieve this, every time an application makes a request, the application proxy intercepts the request to the destination system. Afterwards, the proxy initiates a new request to the destination system rather than sending the application's original request. When the destination system responds back to the application proxy, the proxy responds back to the origin application disguised as the destination system. Thus, an application proxy ensures that there is never a direct interaction between the origin application and the destination system.

Application proxies can have poor performance. Since an application proxy has to initiate a second connection to the destination system, it requires twice as many connections to complete its interaction, which may cause a scalability problem.

5 Network Intrusion Detection System (NIDS)

The idea behind a NIDS is to passively monitor network traffic for signs of an attack. It maintains a table of all active connections with a state for each, where it can look for partial matches to signs of attacks. It is an example of a Man-on-the-Side defensive measure as it doesn't actively interfere with the network traffic.

When it sees a new packet which is not associated with any known connection, it will create a new entry in the table for it. NIDS can be set up on either the host, the network, or both.

5.0.1 Network based detection

In a Network-based Detection Systems, the monitoring is done on a central machine which connects a local network to the rest of the internet. It makes use of network and application level information form the IP packets to detect malicious activity. Since the detection is done on the boundary of the local network, all network traffic can be monitored in one place, allowing for centralized management of network defense. Therefore, we don't need to make modifications to end hosts or trust their ability to detect issues.

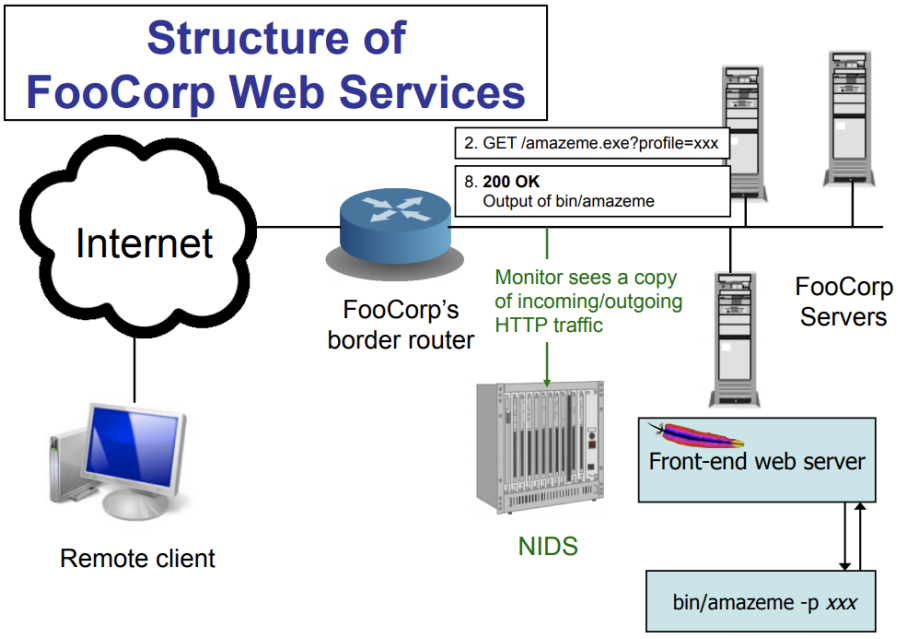

Figure 4: An example of a network based monitoring on a network.

However, this method can quickly become very expensive. When large amounts of traffic pass through, it can becomes costly very quickly. Besides,

Network-based Detection Systems suffers from the issue of imperfect observability where it may not necessarily know whether the end hosts receive the same requests that it receives. A TTL evasion attack will take advantage of this vulnerability of the system and will successfully be able to evade Network-based Detection Systems.

In addition, Network-based Detection Systems faces the possibility of having incomplete analysis techniques. Incomplete analysis is the issue where because the same message can be sent in multiple different formats and encodings, a Network-based Detection Systems may not have accounted for all encodings or formats. An attacker can still evade Network-based Detection Systems by using character escaping techniques like hex escapes and URL encodings which will evade the threat analysis if Network-based Detection Systems are not looking for such a sequence. For example, an attacker can use "%2e%2e%2f%2e%2e%2f" instead of "../../" in order to access a file up the directory tree when the Network-based Detection Systems are trying to prevent parent directory accesses. Another way to take advantage of incomplete analysis is to make use of Linux filesystem artifacts like "..///.///..////passwords.txt" when Network-based Detection Systems are looking for requests trying to access the "passwords.txt" file in the parent of the parent of the current directory by matching "../../passwords.txt" in the request. In principle, a careful implementation that accounts for all possible forms of a malicious string should be able prevent such evasions.

Another issue is that there are many request string that may trigger false positives. For example requests with ../../ may be malicious, but it is also seen in legitimate requests. There are also too many rules and exceptions to handle, for instance the path /etc/password may not exist on all systems and a special rule would be required for each system on the network. It is also impossible to handle all encoding and semantic meanings.

If network traffic is encrypted, with for example HTTPS, then the Network-based Detection Systems would need access to decryption keys, TLS key for HTTPS, in order to be able to detect malicious activity. Providing encryption keys to Network-based Detection Systems can increase your attack surface and hence, such a step must be weighed against its positives.

5.0.2 Host based detection

Host-based Detection Systems are capable of monitoring and analyzing the internals of a computing system as well as the network packets on its network interfaces, similar to the way a Network-based Detection Systems operates. It looks for potentially malicious traffic within a network.

A common tool used to scan requests is arpwatch. Since host based detection can only scan traffic that is traveling within the server, it cannot protect the other end system. So if the end system is compromised and send legitimate requests host based detection would not be triggered.

Host-based Detection Systems solve most of the disadvantages that Network-based Detection Systems have. Host based monitoring doesn't need to intercept

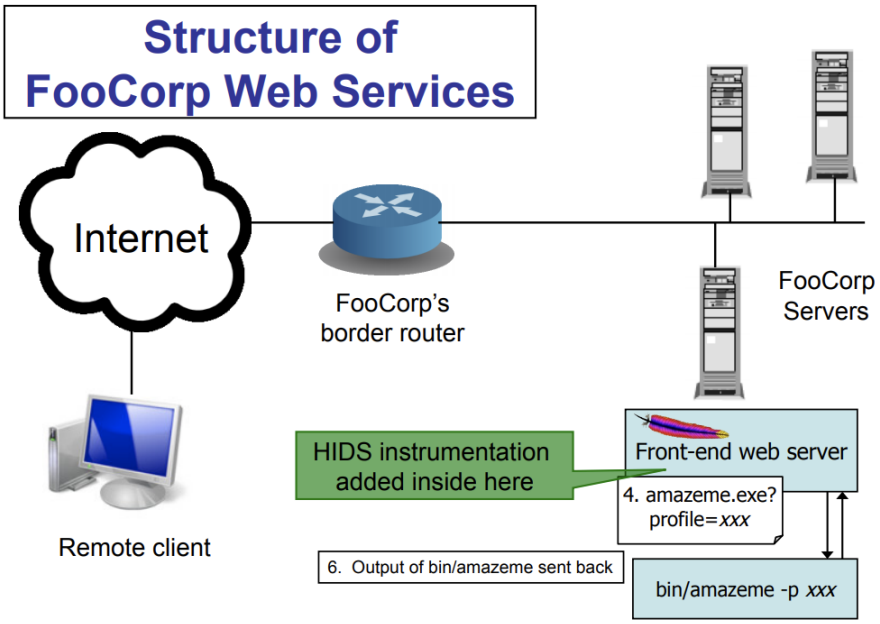

Figure 5: An example of a host based monitoring in a network.

encrypted traffic (HTTPS), and they have understandings of URL semantics, which prevents evasion attacks such as '%2e%2e%2f%2e%2e%2f'. Yet this does not stop everything, as an attacker can still gain access through UNIX filename semantics. Regardless of the protection NIDS provide, careful implementations are still required to prevent most attacks and prevent false positives.

As Host-based Detection Systems are based on each server, it gets progressively more expensive the more servers there are, as each server needs to have code added to them in order to implement the IDS. Therefore, Host-based Detection Systems are more favorable when there are few servers, yet forces the opposite problem where end hosts must be modified and whether or not they can properly detect their issues.

Another issue that is resolved is the fact that Host-based Detection Systems do not require the ability to understand encrypted traffic. Therefore there are no potential security vulnerabilities involving sharing TLS keys. Yet despite this, Host-based Detection Systems only really help with detecting web server attacks, and don't really help with other end systems.

5.0.3 Log analysis

Log analysis: run scripts to analyze system log files (e.g., every night, hour, etc.). One of the benefits of log analysis is that it's cheap since the servers already have logging facilities, and escaping issues will not be a problem because logging

is done by the server.

The disadvantage with this approach is that wouldn't be able to block active attacks, it would only be able to detect attacks afterwards. Several types of malware hide by modifying the log files, which is also a problem.

One example of log analysis is **fail2ban**, a utility that scans log files and blocks IPs that shows signs of malicious intent (such as too many wrong password trials when trying to SSH into a server). Some of the problems with this utility are that the filters are complicated regular expressions, you can accidentally block your self and other users.

5.1 Evaluating efficiency and accuracy

The problems with these intrusion detectors are that it will be inconvenient if the program keeps alerting about every minor/non problem (False positive signals) or if it fails to alert about a real problem (False negative signals).

Detector accuracy is often addressed in terms of False Positive (FP) and False Negative (FN) rates. If I is the event of an instance of intrusive behavor and A is the event of detector generating an alarm, then the FP rate is defined as the probability such that the detector generated an alarm given an instance of non-intrusive behavior (P[A—¬I]) and the FN rate is defined as the probability that the detector did not generate an alarm given an instance of intrusive behavior (P[¬A—I]). The art of a good detector is achieving effective balance between FP and FN rate.

The advantages of having lower FP rate or having lower FN rate depends on several factors. If we look at it from a cost of time perspective, FP's can waste an engineers time, and higher FN's can lead to a huge clean up fee afterwards. It also depends on the rate in which an attack occurs (eg: laptop vs Google's servers).

6 Vulnerability Scanning

Rather than taking a passive defense approach, vulnerability scanning takes an active defense approach by launching attacks on yourself. The goal of vulnerability scanning is to find vulnerabilities by probing internal systems with a range of attacks. If any vulnerabilities are discovered then they must be immediately patched or fixed. Typically this falls under the responsibility of the IT administrators and security team for a given organization.

Vulnerability scans generally look for known vulnerabilities and weaknesses, and there are many versions of specialized scanning software commercially available. This differs from penetration testing as this originates internally and targets attack surfaces with known security holes or exploits. In practice, this approach is prudent and widely used.

Vulnerability scanning can be very beneficial if you are using an efficient and capable scanning tool that allow you to find vulnerabilities. It is an accurate way of finding common issues and loop holes in defense systems. Additionally, having

found the issues before they are exploited by an attacker, you can proactively arrange to fix those issues. However, carrying out a thorough scan of one's systems to be able to ensure that no issue is left unexplored takes a lot of work to set up. It is also not very useful to scan a network for vulnerabilities if the network system cannot be modified to fix the problem. Besides, some attacks that are carried out for scanning purposes may have adverse affects and can potentially disrupt the network system.

7 Honeypots

In the words of Sun Tzu, "know the enemy". With honeypots, we can deploy a sacrificial system that has no operational purpose - think of it like a mouse trap.

Honeypots should appear like a vital part of your system architecture, but will have no actual real purpose for existing — it'll be bait for your enemies.

Since we're creating a system that is seemingly fully operational and filled with activity, attackers will assume it is a vulnerable target and try to exploit it. However, since this entire system only exists as a trap, we can assume that *any* attempt to access this system is malicious.

Provides an opportunity to:

- Identify intruders
- Study what they're up to*
- Divert them from legitimate targets

* - Instead of immediately locking out the attack, we can tag their IP / any other identifying information to ensure that the attacker cannot access the rest of the server.

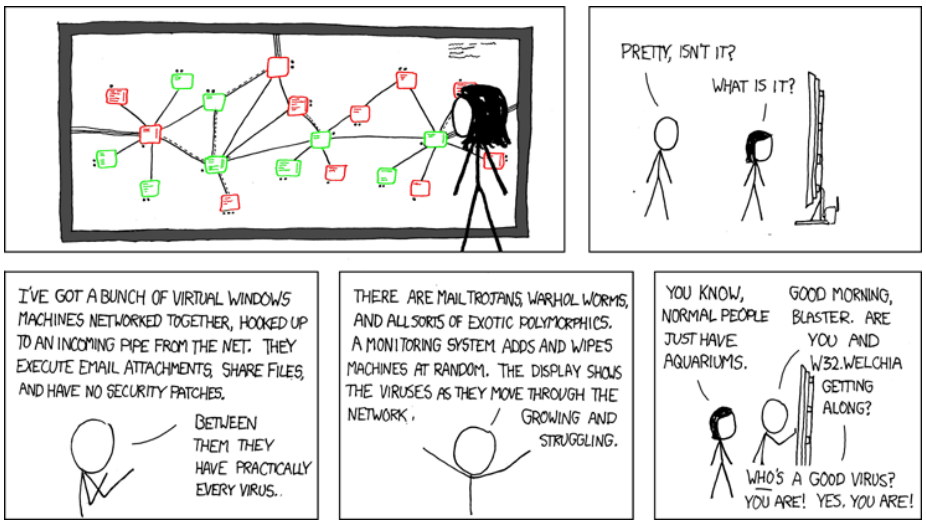

Figure 6: While a joke, this XKCD highlights the benefits of learning from your attackers with honeypots.