CS5430 – System Security Spring 2018

Lecture 25: Network-based Security

April 30, 2018 *Instructor: Eleanor Birrell*

1 Remote Adversaries

In our discussion of threat models, we distinguished between adversaries that have physical or local access to the system and adversaries that only have remote access to the system. But in many cases—such as when the application is running on a server or when targeted users are members of a privileged organization—the adversary has only remote access to the targeted system. Today we will take a closer look at the types of attacks that can be deployed by a remote adversary and the techniques available for defending against remote attacks.

Both local and remote attacks take advantage of the fact that bugs and vulnerabilities are common, and even when a vulnerability is publicly known (and a patch is available) many users are slow to upgrade their systems, leaving a window of vulnerability for a motivated attacker. In previous weeks, we have discussed various ways in which adversaries can exploit vulnerabilities to compromise systems and various techniques for blocking or mitigating such attacks. However, we have generally assumed that the adversary somehow can discover what code is running on a system (and what its vulnerabilities are) and can access that system. A local adversary might learn about vulnerable programs by running `ps`, by inspecting a GUI, or by measuring physical signals. Remote attackers have two general options for bypass this problem, depending on the goal of the attack.

1.1 Random Scanning

In some cases, attackers don't care which principal(s) get compromised as long as they gain control of some machines; in this case, the attackers typically select random IP addresses, contact whichever port the vulnerable service responds on, and check whether a response comes back from a version of the service that is vulnerable to their exploit, and attack all vulnerable systems. In this case, detection requires coordination between independent, randomly selected parties. Even when successful and is thus unlikely (and if it does happen, there will probably be other available targets).

Random scanning for vulnerable machines is a common approach for untargeted attacks. The Mirai botnet, for example, spreads by generating a sequence of random IP addresses and attempting to connect to port 23 on each of those machines; if a connection is established, it tries a hard-coded sequence of authentication credentials containing common usernames and passwords (e.g., ⟨admin, password⟩ and

⟨root,123456⟩) and known defaults (e.g., ⟨root, ikwb⟩, the default for Toshiba security cameras and ⟨admin, 1111⟩, the default for Xerox printers). The CloudPets data leak last month appears to have been part of a ransomware scheme in which hackers scanned random IP addresses on port 27017 (the default MongoDB port) to find Internet-facing MongoDB installations with no authentication. Data contained in vulnerable installations was deleted and replaced with databases with names like "PLEASE_READ" and "PWNED_SECURE_YOUR_STUFF_SILLY"; stolen data could be ransomed in exchange for bitcoin.

1.2 Port Scanning

In other cases, attackers target a particular victim. In this case, the attack will fail (and waste resources) if it exploits a vulnerability in a program that is not running or not accessible on the target system. Worse, from the attacker's point of view, failed attempts might be detected, risking further loss of resources and/or legal action. It is therefore desirable to work out a plan of attack without alerting the target. A common first step for the attacker is therefore to determine what programs are running on the target machine. Whereas a local attacker might determine this information by running `ps`, a remote attacker determines this information by *port scanning*.

A port scanner is a program that, for a particular IP address, reports which ports respond to requests and any available information about the daemon handling each port. A common technique is to issue TCP syn packets to each port and see which return a syn-ack packet. (The behavior when a port is not opened is varied—if the machine does not exist there is typically an ICMP response from a router, if the machine exists but the port is closed the response is typically a TCP reset packet, if the syn request is intercepted there is typically no response—but in all cases an open port is easily identified). From a port scan, an adversary can learn which standard services are running and responding, what operating system is installed, what applications (and versions) are present, and the general topology of the network. Example results from a port scan are shown in Figure 1. With this information, an attacker can determine which vulnerabilities are present and available on the target machine and design an attack accordingly. Port scanners are readily available both open source (e.g., nmap) and commercially, but the legality of scanning machines without permission is complex and under debate (and doing so might be against your ISPs terms of service).

In the absence of any network defenses, an attacker who can detect the presence (and responsiveness) of an application with known vulnerabilities can attack the system remotely by sending packets that trigger the corresponding exploit. This renders machines vulnerable not just to adversaries with local access but to any adversary with network access.

```
Starting Nmap 7.40 ( https://nmap.org ) at 2017-03-18 21:43 EDT
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.12s latency).
Other addresses for scanme.nmap.org (not scanned):
           2600:3c01::f03c:91ff:fe18:bb2f
Not shown:  993 closed ports

PORT       STATE  SERVICE     VERSION
21/tcp     open   ftp
22/tcp     open   ssh         OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.8
                              (Ubuntu Linux;protocol 2.0)
80/tcp     open   http        Apache httpd 2.4.7 ((Ubuntu))
554/tcp    open   rtsp
7070/tcp   open   realserver
9929/tcp   open   nping-echo  Nping echo
31337/tcp  open   Elite

Device type:  general purpose
Running (JUST GUESSING): Linux 3.X (85%)
OS CPE: cpe:/o:linux:linux_kernel:3.13
Aggressive OS guesses:  Linux 3.13 (85%)
No exact OS matches for host (test conditions non-ideal).
Network Distance:  13 hops
Service Info:  OS: Linux; CPE: cpe:/o:linux:linux_kernel

Nmap done:  1 IP address (1 host up) scanned in 20.31 seconds
```

Figure 1: Example Port Scan

2 Firewalls

Today billions of computers (and other devices) are connected to the Internet. This introduces lots of opportunities for remote attackers to exploit vulnerabilities remotely, whether their attack is targeted or untargeted. The standard approach to mitigating remote attacks is to deploy a *firewall*.

The term firewall is somewhat overloaded: it can mean either a program that monitors and filters network traffic (sometimes called a *personal firewall*) or it can mean a dedicated machine running a firewall program that mediates all network traffic to and from a subnetwork. There are several types of firewalls, differing in the types of filtering they do (and the corresponding overhead) and on the types of attacks they can detect and prevent.

Packet Filtering. The simplest form of firewall is a packet filter. A packet filter inspects the IP header of each incoming packet and filters packets according to a predefined security policy—a set of rules stating which source ip, source port, destination ip, destination port, protocol, or combinations thereof are allowed (or disallowed). Packet filters are simple, efficient, and effective; they can be configured to drop all packets from untrusted sources or eliminate any packets destined for services (ports) that the system or subnetwork does not expect to use. The primary shortcoming of packet filtering is its simplicity; packet filtering is both stateless and independent of the packet contents. Both of these limitations impose limitations on the defenses that can be implemented by a packet filter. A packet filter cannot, for example, detect or block a port scan.

Stateful Inspection. Like packet filters, stateful inspection firewalls inspect the header in order to decide whether to filter a packet; stateful inspection firewalls also maintain state information that can be used to make this decision. This state can be used to improve performance (authorization decisions can be made for the first packet in a connection and then remembered). It can also enable detection and mitigation of a larger class of attacks.

One application of stateful inspection firewalls is to detect and block unauthorized port scans. For example, a firewall might block streams that access many different ports, streams with many failed access attempts, or streams that access unusual destinations (using an *anomaly score*).

Another application can be used to mitigate actual attacks. Attackers, in an effort to bypass content-based blacklisting of known attacks, often break their attack up into a series of many (short) packets). However, a stateful inspection firewall can detect and block streams that send many short packets.

Deep-Packet Inspection. Deep-packet inspection cover a sophisticated class of firewalls that make authorization decision on the basis of not only the packet headers but also the packet contents. Firewalls that implement deep-packet inspection might simulate packet behavior, compare packet payloads to a blacklist of known attacks, or enforce copyright restrictions (or general censorship) on user content.

On approach to developing content-dependent firewalls is to leverage existing intrusion detection systems like Snort. Snort is an open-source intrusion prevention system capable of real-time traffic analysis, logging, and packet filtering. It operates by defining a set of rules. Each rule specifies a set of matching packets (this can depend on both the packet header and the packet contents) and an action to take (drop, log, or alert). Snort also distributes packages of rules that match known vulnerabilities in common software, including operating systems, browsers, plugins, and databases.

3 Denial of Service

Denial of service attacks target the availability of resources. There are many ways in which a resource can be rendered unavailable, including crashing a program, deleting (or encrypting) files, and causing hardware failures. Today, we will focus on a class of remote denial of service attacks that focus on overloading network resources, thereby rendering a service or resource unavailable; when such attacks are implemented using many machines (as is usually the case), they are known as *distributed denial of service (DDoS)* attacks. DDoS attacks are typically enacted by a botnet—a collection of machines that have been compromised and now respond to commands from control servers—and are available as a service for hire.

Ping Flood. The simplest form of DDoS attack, a ping flood issues ping requests to the target. If the attacker has sufficient resources to saturate the target's bandwidth, the target will become unavailable to other users.

Syn Flood. A syn flood is a form of DDoS attack that takes advantage of how TCP handles sessions. Recall that a session is established with a three-way TCP handshake: the source sends a `SYN` (synchronize) packet to the destination, the destination sends back a `SYN+ACK` packet, and the source completes the handshake by sending an `ACK` (acknowledge). The destination maintains a `SYN_RECV` queue that keeps track handshakes in progress (those for which it has sent a `SYN+ACK` response but not yet received the final `ACK`). The default queue size on most modern machines is 1024 (machines with low memory have smaller queues); pending handshakes typically time out after minutes. A `SYN` flood proceeds by sending `SYN` packets but never completing the TCP handshake, filling up the queue and leaving the target unable to respond to additional incoming requests.

SYN floods are so common that TCP supports a special-purpose defense designed to mitigate this threat: SYN cookies. The idea is to eliminate the need to store pending SYN packets by encoding the necessary information in the sequence number using a pseudorandom function (more specifically, and HMAC). This prevents the `SYN_RECV` queue from filling up, mitigating the thread of SYN floods. However, SYN cookies do not support all of the options available in the full TCP spec and are therefore officially recommended only as a last resort.

Reflection Attacks. Reflection attacks are a class of DDoS attacks in which the attacker does not send packets directly to the target but instead sends requests to third-parties with spoofed source IPs, causing the third parties—the *reflectors*—to send all their responses to the target. Reflection attacks can be deployed in a variety of forms, including ping floods, syn floods, and DNS attacks. DNS attacks work by sending the spoofed requests to DNS servers; a key feature of DNS attacks is that they can amplify the DDoS attack—that is, cause the target to receive more data that

is sent by the source botnet—by using the EDNS0 DNS protocol extension (which allows for large DNS messages) or using the cryptographic feature of the DNS security extension (DNSSEC) to increase message size.

4 Mitigating Denial of Service Attacks

Mitigating denial of service attacks is a complex and evolving art. Several companies (Akamai, CloudFlare, Google Shield) offer DDoS mitigation as a service, and their precise techniques are generally a closely-guarded trade secret. DDoS-mitigation strategies include content distribution (increasing the number of available replicas), distributed scrubbing (a distributed reverse proxy identifies and drops malformed or suspicious traffic), traffic shaping (buffering and rate-limiting incoming traffic), and egress filtering (preventing outgoing traffic from being used in a DDoS attack). As a general rule, DDoS mitigation depends on deploying extensive defensive resources and can be bypassed by a sufficiently determined and resourceful adversary.