Conclusions

6.1600 Course Staff

Fall 2023

To conclude, we will explore five case studies that exemplify what we have covered so far.

Disclaimer: This set of notes is a work in progress. It may have errors and be missing citations. It is certainly incomplete. Please let the staff know of any errors that you find.

1 *Authentication: OPM Hack*

In order to get security clearance to view classified documents, it is necessary to fill out a form called an SF86 that covers all kinds of personal details from relationships and mental health to drug use and finances—some of the most sensitive data there is. Many, many people have filled these out—around 2.8 million people have some level of current security clearance. The goal of this invasive background check was to understand people's exposure to blackmailing.

These records are stored in a database at the Office of Personnel Management. In 2015, the OPM announced that 20 million of these records have been exposed. This was a big problem for the US Government. Not only did it expose exactly how to blackmail every person with security clearance, but also records of CIA employees were not stored in this OPM database—this meant that if you knew someone had a security clearance but they were not in the database, you had an idea that they might be a CIA agent.

1.1 *How it Happened*

Learn Contractor Credentials. First, the attacker somehow got a contractor's credentials. This could have been through phishing or some other means. The system did not use two-factor authentication, and only around 1% of OPM users used smart cards. Importantly, the contractor did not need to have many privileges on the system—perhaps only enough permissions to log in.

Compromise Root Account. Then the attacker likely compromised the root account on a local machine. Given that these systems were old, this was likely easy—old version of Windows were not particularly careful about protecting access to the root account. One way to do this was by scheduling a job for the future. Windows allowed unprivileged users to schedule jobs that would run as the system user, so a command like `at 16:05 /interactive "cmd.exe"` would open a command prompt as the system user at 4:05 PM.

Learn Administrator Credentials. Even with access to the root account on a local machine, the attacker needed to learn credentials to be able to log into one of the machines with access to the database. Windows did not store the password of the currently logged in user, but it did store the *hash* of the password and uses that to authenticate to the server. This means that the client does not need the cleartext password to log in!

As we have covered many times, passwords are terrible for authentication. Signatures, as used with standards like FIDO2/WebAuthn, provide much stronger security and should be used whenever possible.

2 *Transport Security: POODLE*

TLS is used all over the web. Many servers, from data centers to embedded hardware devices, like doorbells, implement TLS. Some of these servers take a long time to upgrade, so even if a client supports the latest version of TLS (e.g., TLS 1.3), it might be unable to use TLS 1.3 to communicate with some servers. To handle this situation, TLS implements version negotiation; the client and server should agree to use the most recent version of TLS supported by both sides. However, the way this negotiation was implemented was subtly insecure.

When an old server receives a message from a new client speaking a new version of TLS that the server does not support, the server might reply back with garbage data. This could be due to a bug in the server implementation, which was never found because the developers didn't think in advance to test against various ways that future clients sending messages from a future version of TLS. Despite the fact that this is a server bug, users want to still be able to communicate with this device; after all, communicating with the device worked fine before the client upgraded to support a new version of TLS. As a result, clients that receive garbage data in response to their TLS connection attempt will try to downgrade the version of TLS that they use to connect in their next attempt. However, the "garbage" that the client is reacting to was not authenticated. Therefore, an adversary could inject garbage to force a client to downgrade to an older TLS/SSL version.

Old versions of TLS/SSL have well-known weaknesses in their cryptographic protocol. In particular, SSL 3.0 computed the MAC of the message (for authentication) before encrypting the message (MAC-then-encrypt), rather than the encrypt-then-MAC approach that we discussed in the authenticated encryption lecture. This allowed the attacker to send corrupted ciphertexts and see how the client or server respond to them. In combination with MAC-then-encrypt, SSL 3.0 also used a particular encryption construction, called cipher-block-chaining (CBC) mode, whereby changing bits in one part of the ciphertext caused corresponding changes to another part of the plaintext. Finally, the plaintext payload included padding, constructed with a well-known scheme, which was checked after decryption to make sure it was not corrupted; if the padding was corrupted, the connection was terminated. However, this gives a signal to the adversary as to whether their corrupted ciphertext happens to match the expected padding or not. We saw the CBC padding oracle attack earlier in the lecture on authenticated encryption.

The POODLE attack was a combination of these two weaknesses (downgrade attack and CBC padding oracle), together with running adversary-supplied code in Javascript in the victim user's web browser, as a way of sending partially-adversary-chosen requests over the TLS/SSL connection. The result is that the adversary can recover other data sent to the server over the same connection, such as the victim's cookie.

3 *Platform Security: Sony PS3 Hack*

The Sony PS3 originally could boot Linux and Windows. Since the hardware was subsidized by the games that they sold, PS3s were cheaper than comparable PCs, and for that reason PS3s were a popular option for a cheap PC. In a software update, Sony disabled the ability to run a custom operating system. Like the iPhone and other systems we discussed, PS3s then used secure boot to ensure that they only boot Sony-signed operating systems.

Sony used EC-DSA for their signatures, which resulted in signatures along the lines of:

$$\sigma = (g^r, r + H(\mathsf{pk}||g^r||m) \cdot \mathsf{sk}) \pmod q$$
$$\sigma' = (g^r, r + H(\mathsf{pk}||g^r||m') \cdot \mathsf{sk}) \pmod q$$

In EC-DSA, r is supposed to be a long random number, serving as a nonce. However, Sony re-used these nonces in pairs of signatures. This meant that their signatures revealed their secret key! This allowed others to sign their own operating systems.

Importantly, Sony had a plan for updating the PS3 firmware that allowed them to ship a fix for this attack. However, attackers quickly found flaws in every other update they shipped—it is very hard to secure a device that the attacker has unconstrained access to.

4 *Software Security: WannaCry Ransomware*

Ransomware is a type of malware that encrypt important-looking files on the infected system and demands a payment in Bitcoin to decrypt the files. This is inconvenient and upsetting for personal computers, but for enterprise computer systems this can cause huge monetary losses. For hospital systems, this can even lead to loss of life. This WannaCry randomware infected hundreds of thousands of computers and caused billions of dollars of damange, but did not make much money due to bad payment systems and slow decryption.

The bugs used to enable this were part of an exploit developed and kept secret by the NSA called EternalBlue. It took advantage of several C bugs, including an invalid cast, a parser bug, and an allocation bug, to eventually achieve remote code execution over the network on a Windows system.

The NSA intended to keep these exploits to themselves, and thus did not inform Microsoft (or anyone else) about the bugs in their software. However, an NSA contractor took terabytes of NSA data home with him, including this exploit. On his home computer, he ran Kaspersky antivirus. Importantly, Kaspersky sends suspicious files home for analysis. Wall Street Journal reported that this was likely how the exploit leaked and became a part of malware.

In order to spread to many computers, WannaCry looked something like the following:

1. Connect to a website at a random-looking address and exit if it succeeds. Security teams would often analyze software that they thought may include malware by running it in a VM, allowing network requests to succeed, and watching what happens. This random-looking domain did not really exist, so this may have been a way to try to detect when the software is running in a VM and make it behave "normally“.

2. Install Tor and connect to command-and-contral infrastructure.

3. Encrypt all files with a fixed set of extensions with RSA and AES.

4. Demand a ransome to be paid to one of four static Bitcoin addresses.

5. Spread itself by trying to perform the exploit on all IPs in the local network.

When designing a system, it is prudent to have the system as simple as possible, since less software leads to fewer bugs. And of course, any bug is a security bug—each of the bugs used in the EternalBlue exploit did not look like a security bug, but the combination of them allowed for a powerful exploit.

5 *Privacy: US Census*

The US Census, performed every decade, collects data used to allocate seats in the House of Representatives and by many researchers. The Census Bureau is mandated to make this information public, but is forbidden by law from publishing any data that allows individuals to be identified.

In the 2020 census, the bureau used differential privacy to protect released data from de-identification. However, they used $\epsilon = 19.61$ to avoid adding so much noise that the data lost its utility. This meant that if the probability of some event happening to an individual without the release of these data was p, the release of these data with $\epsilon = 19.61$ was guaranteed to make the probility at most $\approx e^{19.61} p$, or around a million times the original probability.