In this section, will discuss about:
- Kernel Exploits
- Service Exploits
- Registry Exploits
- Passwords
- Scheduled Tasks
- Insecure GUI Apps
- Startup Apps
- Installed Applications
- Hot Potato
- Token Impersonation
- Port Forwarding

---

## Kernel Exploits:

- 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 kernel version ( `systeminfo` )
  2. Find matching exploits (Google, ExploitDB, GitHub)
  3. Compile and run.

⚠ **Beware, Kernel exploits can often be unstable and may be one-shot or cause a system crash**

GET TOOL - 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

---

## Service Exploits:

- Services are simply programs that run in the background, accept 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 (eg; SERVICE_QUERY_CONFIG, SERVICE_QUERY_STATUS)
  - Some may be useful (eg; SERVICE_STOP, SERVICE_START)
  - Some are dangerous (eg; SERVICE_CHANGE_CONFIG, SERVICE_ALL_ACCESS)

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

### Unquoted Service Path:

- Executables in Windows can be run without using their extension (eg; "whoami.exe" can be run by just typing `whoami`)
- Some executables take arguments, separated by spaces, (eg; someprog.exe arg1 arg2 arg 3...)
- This behavior leads to ambiguity when using absolute paths that are unquoted and contain spaces

Consider the following unquoted path:
`C:\Program Files\Some Dir\SomeProgram.exe`

To us, this obviously runs SomeProgram.exe but 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.

### 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

### 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

### 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
- 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 are difficult, and often the entire process is very manual

---

## Registry Exploits:

### 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

### 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 (eg; admin) privileges
- If this is the case, we can generate a malicious MSI file which contains a reverse shell

**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**

---

## Passwords:

- Even system administrators reuse 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
- 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

## Saved Credentials:

- 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

## Configuration Files:

- Some administrators will leave configuration 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

When 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`

## SAM: (Windows Security Manager)

- Windows stores password hashes in the Windows Security Manager
- 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

**For NTLM hashes with `31d6fe0d16ae931b73c59d7e0c089c0` , it is likely an empty password or disabled**

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

Example of command:
`pth-winexe -U 'admin%hash:hash' //127.0.0.1 cmd.exe` (To spawn a shell / elevated shell)

---

Scheduled Tasks:

- Windows can be configured to run tasks at specific times, periodically (eg, every 5 minutes) or when triggered by some event (eg; 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

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

---

## Insecure GUI Apps:

- On some (older) versions of Windows, users could be granted the permission to run certain GUI apps within 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
- 0xtib3rius call this the "Citrix Method" because it uses many of the same techniques used to break out of Citrix environments

---

## 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

---

## Installed Applications:

> **Using everything you have learned so far in the course, it should be no problem identifying exploits with currently installed applications and using their exploit-db entry to escalate your privileges**

---

## 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

## Token Impersonation:

### Service Accounts:

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

### SeImpersonate / SeAssignPrimaryToken:

- Service accounts are generally configured with these two privileges
- They allow the account to impersonate the access tokens of other users (including the SYSTEM user)
- Any user with these privileges can run the token impersonation exploits

### 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
- Get the exploit here: https://github.com/ohpe/juicy-potato

### Rogue Potato:

- Latest of the "Potato" exploits
- GitHub: https://github.com/antonioCoco/RoguePotato
- Blog: https://decoder.cloud/2020/05/11/no-more-juicypotato-old-story-welcome-roguepotato/
- Compiled Exploit: https://github.com/antonioCoco/RoguePotato/releases

## PrintSpoofer:

- PrintSpoofer is an exploit that targets the Print Spooler service
  Github: https://github.com/itm4n/PrintSpoofer
  Blog: https://itm4n.github.io/printspoofer-abusing-impersonate-privileges/

---

## 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)

---

## Golden Trove - Strategy

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
   (eg; https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Windows%20-%20Privilege%20Escalation.mdcalation/)

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
- Have a quick look around for files in your user's home directory and other common locations
  (eg; `C:\` , and `C:\Program Files` )
- Read through interesting files that you find, as they may contain useful information that could help escalate privileges
- Try things that don't have many steps first, eg; 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

- If you still don't have 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 (last resort)

**Don't Panic**

Privilege Escalation is tricky
Practice makes perfect
Remember: in a 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 !

---

Extras:
getsystem - (Named Pipes & Token Duplication)

Access Tokens:
Access token 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 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?
  The source code for the getsystem command can be found here:
  [https://courses.tib3rius.com/courses/windows-privilege-escalation-for-oscp-beyond/lectures/43490902](https://courses.tib3rius.com/courses/windows-privilege-escalation-for-oscp-beyond/lectures/43490902)
- 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:

- 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 SeTakeOwnsershipPrivilege 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)

---

Back - Tib3rius' Windows Privilege Escalation - Introduction (Harvested in 2025)