

From Zero to Hero “Exploitation 101”

Objectives

In today's hands-on labs, you will perform the following four labs:

Part #1 – Preparations

Part #2 – Port Scanning for Running Applications

Part #3 – Fuzzing the Network Service using SPIKE - DIY

Part #4 – Exploit Development (Writing a PoC)

Part #5 – Connecting to the Exploited Box

Part #6 – Reflection on the lab

Overview

Before we start our penetration test, we need to gather as much information as we can about the target (assume we did that). This lab is going to the next phase of our attack, the port scanning phase. This phase aims to gather information about open ports, running services, services versions, and the operating systems running. The final report of this phase will help us move to the next phase of our attack, so don't forget to prepare a final report of your discovery.

Requirements:

1. Two machines or virtual machines are needed: KALI and your Windows 10.
2. Vulnserver Application, found [here](#).

Part #1 – Preparations

Let's start by finding the IP Address of the KALI and Windows Machine (*this could be skipped since you know it!*).

1. Open a Terminal window by going to **Applications** ➤ **Accessories** ➤ **Terminal**.
2. In the Terminal window, execute the **ifconfig** command. Make a note of your IP address for later reference.
3. On the Windows machine, click **Start** ➤ **Run**. In the Run box, enter **cmd.exe** (cmd alone is enough) and press the Enter key. In the Command Prompt window, enter the **ipconfig** command and press the Enter key.

Setting Your Windows 10 Machine's Firewall to Allow All Incoming Connections

4. In your Windows 10 target machine, disable the "Windows Firewall".

Ensuring that You Can Reach the Target

5. In the Terminal window, after the # prompt, enter the command below, then press the Enter key (don't forget to replace the IP address with your Windows IP address):
ping 192.168.8.10
6. You should see lines starting "**64 bytes from...**". Press Ctrl+C to stop the pinging.

Task #1.1: What IP address and subnet mask your KALI Linux system using now? (*Remember it!*)

7. If you don't see any replies, your virtual machine is not able to reach your Windows target. You need to be able to communicate with the Windows target to proceed with this lab. Try troubleshooting it.

Starting our Vulnerable Network Application (*this cannot be skipped*)

8. Download the file named "[vulnserver.zip](#)". Extract the file, and then enter inside the directory and double click on the application "vulnserver.exe" to start our vulnerable network application. You can find the application within our shared course drive.

Note: The application "**vulnserver.exe**" will serve as our vulnerable application that we need to exploit using our KALI Linux machine.

Part #2 – Port Scanning for Running Applications

Port Scanning Your Windows Machine with nmap

1. Currently suppose you don't know on what port it is listening. So we want to discover the port that the "**vulnserver.exe**" is running on. From our Linux machine inside the Terminal window, after the # prompt, enter this command (Use your windows IP Address):
`# nmap -sS -p 1-10000 <windows-ip-address>`
2. Press Enter to start the scan.
3. When the scan completes, scroll back to see a chart showing the open ports in green text (if you used Zenmap). Your Windows machine should have a number of open ports.
4. A TCP port is supposed to be used for the "**vulnserver.exe**", and it's open, which means we are ready to communicate with it.

Task #2.1:

- A) What are the running TCP services, and what ports are they using?
- B) What is the TCP port used by our vulnserver.exe application?

Part #3 – Fuzzing the Network Service using SPIKE - DIY

Write a SPIKE fuzzer to crash the vulnerable server.

Part #4 – Exploit Development (Writing a PoC)

Exploiting the Application

The vulnerability we are willing to exploit is a stack based buffer overflow in the argument passed to the **TRUN** command of vulnserver.exe.

Checking for a Vulnerability

1. Again from your Linux machine adjust the python exploit template to your needs then run use it as following:

python exploit.py

Note: exploit.py will serve as our exploit.

```
#####
#!/usr/bin/env python
import socket
host = "<windows-ip-address>"
port = <vulnserver-port>
cmd="TRUN ."
offset="A" * USE_A_NUMBER_HERE # cause the crash
shellcode = cmd + offset
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host,port))
data=s.recv(1024)
print "\n" + data
s.send(shellcode)
s.close()
print "Done Sending: ", len(shellcode), " bytes"
#####
```

2. In the Windows machine check if your "**vulnserver.exe**" has crashed after sending a huge number of bytes after the TRUN command. This can be done by adjusting the number used in the template above "**USE_A_NUMBER_HERE**".
3. Restart your test and check if the application crashed.

Task #4.1: How many bytes did you send with the TRUN command?

Move on to the next step...

Checking the Crash with a Debugger (ImmunityDebugger)

4. Now let's start our debugger. Go to start ⑦ All Programs ⑦ Immunity Inc ⑦ Immunity Debugger, and click on the Immunity Debugger icon.
5. By now you should have crashed your "vulnserver.exe". Within the debugger goto File ⑦ Open and navigate to the directory of "vulnserver.exe" and select it. Press F9 to start the application within the Debugger. (Now "vulnserver.exe" is running from within our debugger, we will use it to continue our exploit development)

Checking the Control of EIP

6. Go back to the Linux machine and run the exploit again.

Task #4.2: Check the debugger; Did the application crash? How can you prove that?

7. By now you must have managed to overwrite the EIP register. If you did continue to the next step, if you didn't then adjust the **USE_A_NUMBER_HERE** and restart the application inside the debugger, and run your exploit again until you make sure that EIP is overwritten with four A's.

Checking the Offset to EIP

8. Now that we confirmed EIP is overwritten with 4 A's, we need to calculate the offset to those 4 A's. We can use the Metasploit's **pattern_create** to create a pattern with the size of the buffer that triggered the exploit. This can be done as follows:

```
# pattern_create ????
```

9. Now replace the created pattern instead of the buffer number you used before and restart the process.

Task #4.3: Check what are the 4 bytes that have written over EIP, write them down for our next step



10. Now we're going to use the Metasploit's **pattern_offset** to check the offset that is needed to overwrite EIP. This can be done as following:

```
# pattern_offset "REPLACE_WITH_4_BYTES_FOUND"
```

Note: please adjust the commands `pattern_create` and `pattern_offset` to suite your testing environment, because the location of the `pattern_create` will be different on different KALI versions.

11. Now use that number for the buffer and adjust your exploit to send the EIP 4 bytes as "B" and the rest of the buffer as "C" for example. (Explanation: let's say we needed 1000 bytes to overflow the application, and the offset was 350 bytes. Then use "A" * 350, plus "B" * 4, plus "B" * 646 for the rest of the buffer, which is a total of 1000 bytes).
12. Restart your testing.

Searching for the Address for

13. Since this is a basic BoF and NO memory randomization is applied to the application. We need to find an address that can be used to jump to our payload instead of the four "B" bytes.
14. Goto view  Executable modules, then double click on a module you think is always going to be running with this application.
15. In the Disassembler's pane, right click and goto search for  command.
16. We want to search for the assembly instruction "**JMP ESP**". Write down the found address.
17. Now press F2 on that address to place a breakpoint.
18. Adjust your exploit template to now use the address found instead of the four "B" bytes. Don't forget the little Endian stuff ;)
19. Restart your test, and now your application is supposed to stop on the breakpoint we placed.

Task #4.4: What is the opcode for JMP ESP? Why did we search for a JMP ESP assembly instruction?

Adding the Shellcode

20. Since everything is working as we want, it's time to add our shellcode.
21. Generate your own shellcode for running a message box and a reverse tcp shell.
22. Replace the "C" bytes we have in our exploit template with the shellcode, but before that add some NOP's to make sure when our jump address is run, we're going to jump within the buffer leading to our shellcode.
23. So add some NOP's which is hex "\x90", and then add your shellcode.
24. Now restart your work, and check if the program stopped within the debugger or not. If it did, then you have something wrong and you need to check it. If the debugger shows "running" at the bottom right of the debugger's window, then proceed to the next step.

Part #5 – Connecting to the Exploited Box

If your debugger in the previous steps showed that the application is running even after running our exploit, then it's time to run the application without the debugger.

Connecting to the Windows Box

1. This section depends on the type of shellcode you used.

Task #5.1: How can you connect to the target box (the exploited Windows box)?

2. By now if you are able to connect to that port, you are given a cmd.exe shell on the target. If you didn't then check the previous steps and make sure you've done them correctly.

Task #5.2: How can you prove that you are truly on the Windows box?

3. If all went successfully, then congrats for another successful software exploitation (buffer overflow) lab!

Task #5.3: Now suppose your target is running behind a firewall. Will the approach we did above succeed or not? Why?

Part #6 – Reflection on the lab

Please reflect back on what you learned from this assignment.