LAB 3 – Metasploit e buffer overflow

Sicurezza e Privatezza

The learning objective of this lab is for students to get the basics of exploitation and post-exploitation with Metasploit and a little dive into memory errors with buffer overflow attack.

Students are invited to complete the following exercises by write the exploit setting and usage in a txt file or attach the screenshot that prove the success of the exercise and submit it to SPLab@di.unimi.it, with subject: 2020Lab3NameSurnameUniversityid (ex. 2020Lab3MatteoZoia876271). Message not conforming to these rules will be automatically rejected.

The first five students who will respond correctly to the entire lab will receive a bonus of 0.5pt on the final grade, 0.25pt on Metasploit attacks part and 0.25pt on the buffer overflow part.

Metasploit attacks

Exercise #1: First attack

In this task, we will use the `unix/misc/distcc_exec` exploit to demonstrate the overall process. Make sure that both VM1 (Kali) and VM2 (metasploitable) are booted. In VM1 launch Armitage and find the exploit distcc_exec: `exploit/unix/misc/distcc_exec.`

Determine the available options by double click with Armitage on the selected exploit (or with Metasploit: `msf exploit(distcc_exec) > show options`).

To make the exploit work, you still need to specify the values of related environment variables, e.g., *RHOST*, the target IP address of vulnerable machine (`ifconfig` on the metasploitable). You can leave *PAYLOAD* to the default value or you can use something else (`cmd/unix/reverse`).

Finally, run the exploit by simply running `exploit` in metasploit or launch in Armitage. **For this exercise submit a screenshot of the window with the working exploit (prove that you gain root privilege on the matasploitable machine) and reports all command used.** There is a PDF file to guide you through this exercise.

Exercise #2: Tomcat server attack

In this task you will learn about reconnaissance and scanning ports. Armitage contains a plugin that wraps nmap for us; when an host is scanned, the results are stored within a database for later access. **Run a portscan** with Armitage or throught msf on the metasploitable target (`db_nmap –sV 10.10.15.x`). You can view scanned hosts with `db_hosts`.

Your portscan should reveal that the target is running Apache Tomcat. You can find an exploit for this service named *multi/http/tomcat_mgr_deploy*. Be sure to set your exploit options appropriately. Google for **default credentials for this service**. Use the nmap data to set RPORT. **Use the techniques you learned from exercise 1 to gain remote control. For this task you need to report all your commands, options and a screenshot (to prove that you gain root privilege on the matasploitable machine).**

Buffer overflow

Exercise #1: Modifiable

The following program accepts user-input through the gets function and then looks for a specific value in a local variable named modified. If this value is non-equal to 0, `printf` function is used to show a win message to the user. There is no direct update on "modified" variable in the program flow and it's value is always 0.
The gets function will keep reading from the stdin device until it encounters a newline or `EOF` character. Since this reading loop fails to honor the size of the destination buffer, a classic buffer overflow vulnerability is introduced in the program. **The student has to find this vulnerability and exploit the program es1.c so that it prints the win message to `stdout`. Attach a screenshot of the `stdout` and the payload given as input to the program.**

Compile with canary guard disabled:

```
gcc -fno-stack-protector -z execstack file.c -o file
```

Disable ASLR countermeasure before running:

```
sysctl -w kernel.randomize_va_space=0
```

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

int main(int argc, char **argv)
{
  volatile int modified;
  char buffer[64];

  modified = 0;
  gets(buffer);

  if(modified != 0) {
      printf("you have changed the 'modified' variable\n");
  } else {
      printf("Try again?\n");
  }
}
```

Exercise #2: Flow changer

The following program has a buffer overflow vulnerability, most of the concepts are very similar to the previous exercise, but at this time you need to hijack the regular program flow in order to print the success string. **The student has to find the vulnerability and exploit the program es2.c so that it prints the "you win…" message to `stdout`. Attach a screenshot of the `stdout` and the payload given as input to the program.**

Remember to disable ASLR and stack guard

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

void win()
{
  printf("You win, code flow successfully changed\n");
}

int main(int argc, char **argv)
{
  char buffer[64];

  gets(buffer);
}
```

Exercise #3: Printme

The following program has a buffer overflow vulnerability, most of the concepts are very similar to the previous exercises but at this time there is no function that prints a "You win .." message. **The goal is print the string "you win!" your own, think if you can smash the stack and push your own code. Attach a screenshot of the `stdout` and the payload given as input to the program.**

Remember to disable ASLR and stack guard

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv){
  char buffer[64];
  gets(buffer);
}
```