Computer Security
Coursework Exercise CW3
Buffer Overflow Attacks

The goal of this coursework is to gain practical experience with attacks that exploit buffer overflows. A buffer overflow occurs when a program attempts to write data beyond the bounds of a fixed-length buffer. This can be exploited to alter the flow of the program, and even execute arbitrary code. This vulnerability is due to confusion between storage for data such as buffers, and storage for control such as return addresses: overflowing the data part may allow an attacker to change the return address and allow him to affect the control flow of the program

Here you are given four exploitable programs which are installed with `setuid` root in the VM, in the `~/Documents/CW3/` directory of the cscw3 account. Your challenge will be to identify a vulnerability in each of the programs. You will need to implement an exploit for each of the vulnerable programs with specially crafted malicious arguments, spawning a shell with root access although executed by a unprivileged user (cscw3).

1 Setup

For this exercise we are using a Virtual Machine called Ubuntu32 (how original!) inside VirtualBox. It is accessible from all DICE machines in /group/teaching/cs/, in a compressed virtual appliance format. The directory is an AFS mount so you may need to type the full path into a command line or into the folder GUI to see the files. Tab completion will not work and the containing folders will appear empty.

Important You must run your attacks inside the provided VM. It is very important that your attack programs work in the provided Ubuntu32 VM. This is because, the compiler version, the operating system and the installed libraries will affect the exact location of code on the stack. The VM provides an identical environment to the one in which we will test your code for marking it.

The machine comes as an *.ova file, which includes the virtual hard drive as well as all necessary settings for VirtualBox. VirtualBox can be downloaded at `https://www.virtualbox.org/` and is already installed on all DICE computers. Once VirtualBox is running, select File ↪ Import Appliance and select the file Ubuntu32.ova. The only property in the configuration screen you should change is the location of the virtual hard drive. On DICE computers a home directory account does not have enough quota to store the hard drive, so you will need to store the virtual disk locally on the workstation. The disk takes up about 6G once expanded. Go to the last line Hard Disk Controller ↪ Virtual Disk Image. Change the path from, *e.g.*

```
/afs/inf.ed.ac.uk/user/sXX/sXXXXXX/VirtualBoxVMs/Ubunt32/Ubuntu32-disk1.vmdk
```

to

```
/tmp/sXXXXXX/VirtualBoxVMs/Ubuntu32/Ubuntu32-disk1.vmdk
```

(you may need to create the directory `/tmp/sXXXXXX` first). The import operation can take several minutes and need about 6G of disk space per machine.

On DICE machines the virtual disk has to be stored on the local disk. That means that you will not be able to access the virtual machine on any other DICE computer than the one you did this setup on. If you want to use your virtual machine from another computer you will have to log in to the machine using `ssh -X <computername>`, or copy the disk files to another machine with `scp`; so make sure you remember the name of the computer you used!

We have configured the VM with Ubuntu Linux 16.04 LTS. Ubuntu, as most other systems, uses address space layout randomization (ASLR) to randomize the starting address of the heap and the stack. This makes guessing the exact addresses difficult, and as you will see, guessing addresses is one of the critical steps to exploiting a buffer-overflow. In the Ubuntu32 VM, we have disabled this feature.

Ubuntu32 has user "cscw3" with no admin privileges and no password set. Your attacks must run as "cscw3" and should spawn a command line shell (`/bin/sh`) running as "root". The VM comes with the tools you will need (gcc, gdb, emacs, etc) installed.

To further 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 `setuid` 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`. For this coursework, and in order to really realise how much damage this sort of attacks can do, we use another shell program (the `zsh`), instead of /bin/bash. The preconfigured Ubuntu virtual machines contains a `zsh` installation.

The Shellcode For your attacks you will need a piece of code that launches the shell (a shellcode). This is the code you will need to load into the memory and to which you will make the vulnerable program jump. So what we need is the assembly program corresponding to the following C program:

```
#include <stdio.h>

int main( )
{
    char *name[2];
    name[0] = "/bin/sh";
    name[1] = NULL;
    execve(name[0], name, NULL);
}
```

You can use the following shellcode for your attacks:

```
"\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:  cdql              */
"\xb0\x0b"    /* Line 10: movb  $0x0b,%al   */
"\xcd\x80"    /* Line 11: int   $0x80       */
```

2 The vulnerable programs

You can get the source of the 4 vulnerable programs from the course webpage.

vuln1 This is just for worming up. This program waits for a password (stored in the file password.txt), and if the correct password is entered by the user it prints

```
cscw3@myrto-VirtualBox:~/Documents/CW3$ ./vuln1 "**********"
Correct password

You are now logged in with root access
```

If the entered password is incorrect it prints:

```
cscw3@myrto-VirtualBox:~/Documents/CW3$ ./vuln1 "123456"
Incorrect password
```

This program has a buffer overflow vulnerability. It is possible, without knowing the password, to make the program print

```
cscw3@myrto-VirtualBox:~/Documents/CW3$ ./vuln1 "**********"
Incorrect password

You are now logged in with root access
```

Write an attack program attack1.c that fulls vuln1 into thinking that the user is logged in with root access.

vuln2 This program waits for the user's name (given as argument to the program), copies it to a buffer `buf` of maximum length 100 bytes, and then welcomes the user. This program is vulnerable to a buffer overflow attack. Your attack program attack2.c will input to the program a carefully crafted argument such that it loads onto the stack the shellcode, and then jumps to this shellcode and executes it, yielding a root shell even though the attack was run by an unprivileged user.

vuln3 This program is similar to vuln2, except that the user's input is not copied at the beginning of `buf` but at position `index` in the buffer `buf`, which is also passed as an argument to the program.

The GCC compiler implements the "Stack Guard" security mechanism in order to prevent buffer overflows. When this mechanism is enable it includes a random value generated by the calling function, called a stack canary, and loads it onto the stack just after the previous stack base pointer `%esp`. This mechanism was disabled for vuln2 by compiling it with the command `-fno-stack-protector`. But, vuln3 has been compiled without this command, so "Stack Guard" is enabled to defend vuln3 against buffer overflows by including such a canary. However, vuln3 is still vulnerable to a buffer overflow. In other words, it is possible to bypass the "Stack Guard" mechanism. Your attack program attack3.c will input to the program carefully crafted arguments such that it loads onto the stack the shellcode, and then jumps to this shellcode and executes it, yielding a root shell even though the attack was run by an unprivileged user.

vuln4 This program is exactly the same as vuln3. So it allows you to load the shellcode onto the stack and jump to it. However, the shellcode will not be executed. We have compiled vuln4 with the "ExecShield" mechanism enabled. "ExecShield" essentially does not allow executing any code that is stored on the stack. As a result, the previous buffer overflow attack will not work on vuln4. (To disable "ExecShield" in Ubuntu you can just use the command `-z execstack` when compiling programs). But the program is still vulnerable to a buffer overflow, and you still control the return address of function vuln. So you can mount a `return-to-libc` attack against this program.

You will need the addresses of `system()`, `"/bin/sh"` and of `exit()`. Run vuln4 in GDB, set a break point to `main`, run vuln4 with any valid arguments. GDB will stop the execution at the set break point. You can then ask for these addresses.

```
cscw3@myrto-VirtualBox:~/Documents/CW3$ gdb vuln4
(gdb) break main
Breakpoint 1 at 0x80485af: file vuln4.c, line 15.
(gdb) run "aaa" 0
Starting program: /home/cscw3/Documents/CW3/vuln4 "aaa" 0

Breakpoint 1, main (argc=3, argv=0xbffff0c4) at vuln4.c:15
15   vuln(argv[1], atoi(argv[2]));
(gdb) print system
$1 = {<text variable, no debug info>} 0xb7e42d80 <__libc_system>
(gdb) print exit
$2 = {<text variable, no debug info>} 0xb7e369b0 <__GI_exit>
(gdb) find &system,+9999999,"/bin/sh"
0xb7f63a3f
warning: Unable to access 16000 bytes of target memory at 0xb7fbd7c7, halting search.
1 pattern found.
(gdb)
```

Here the addressed GDB returns are `0xb7e42d80` for `system()`, `0xb7e369b0` for `exit()` and `0xb7f63a3f` for `"/bin/sh"`. Your attack program attack4.c will input to the program carefully crafted arguments such that it mounts a `return-to-libc` attack, yielding a root shell even though the attack was run by an unprivileged user.

Note Your attack against vuln4 is also an attack against vuln3. However, for vuln3 we do not expect a `return-to-libc` attack. But one that loads the shellcode onto the stack and jumps to it. So your attack against vuln3 should not work anymore if the "ExecShield" mechanism is enabled.

3 Submission Instructions

You should turn in four exploit programs and a Makefile to build them. Running `make` from your submission should yield 4 executables (attack1 through attack4). To retrieve your attack programs from the VM you just need to change the settings Devices ↪ Shared Folders ↪ Shared Folders Settings. For this coursework you can only submit electronically using the following command

```
submit cs 3 filename
```

You need to submit by the deadline of December 2nd at 16:00.