Project 1 - Understanding Memory Protection

dvenkatesh7@gatech.edu

TASK 1

PART A

1. We get SIGSEGV when we execute the object code because the program is trying to write to the second page which only has read permission (it is write protected).

2. The mprotect function will set the protection to the address range specified in the parameters (addr <-> addr + len - 1) to the value specified in the PROT parameter. If a process tries to access the memory that doesn't comply to the protections placed, a SIGSEGV signal is generated. If the function executes successfully, the function returns a 0. If an error occurred, -1 is returned and errno is set to indicate the error.
   Since mprotect provides protection at page level, the addr and len values should be a multiple of the page size.
   If the len argument is 1, it will be rounded off to the next multiple of page size as returned by sysconf.

3. The mprotect function provides memory protection at page level. The value of addr should be a multiple of page size and value of len should be next multiple. Sam has to protect the entire page that contains this address range. To protect from overwriting, the PROT parameter should only contain PROT_READ permission.

4. In line 81, the mprotect is providing read and write access to the contents on the first page. This means that data can be read from the page and can be written into the first page.
   In line 106, the mprotect is providing only read access to page 2 thereby restricting write access to the page. Any attempt to write into page 2 will result in a SIGSEGV.
   The various printf statements in the code is reading the data and printing it out to the screen.
   Set of lines from line 89 to 93 is a way to overwriting the data (provided it has write access).

PART B

3. After writing userID in 5th and 6th page we can still perform the read even though it only has write access. This is because we cannot just write without reading. Read access is implicitly provided.

4. I was able to create the buffer with 2 pages and was successfully able to copy 7th and 8th page into it.

5. I was able to successfully copy 6th and 9th page into the previously created buffer. I was able to overwrite the contents as mprotect by default provides both read and write access on pages. (PROT_READ and PROT_WRITE)

TASK 2

PART A

1. In the stack, each function is allocated a separate stack frame. The stack frame is a place where a function stores its return address, parameter values, register values etc. The stack smashing happens when the attacker deliberately overfills the stack thereby overwriting the stack contents with the data the attacker wants. This will change the return addresses stored in the stack and the attacker will be able to go the memory location where the malicious code is present. This is called Stack Smashing.

2. When a return address is stored in the stack frame, a canary value - a randomly chosen integer value - is written just before it when the function begins. If the attacker tries to rewrite the return address using the buffer overflow attack, this canary value will also be overwritten because when the buffer overflow happens, the data is overwritten from lower to higher memory addresses. This will detect that an overflow has occurred, and the program will terminate.

3. This technique makes the stack non-executable. So, if the attacker has injected a shell code on to the stack, he will not be able to execute it as the stack is made non-executable. The NX bit is provided by the hardware. Even though this provides a certain level protection, it does not work against the return-to-libc attack.

4. Address Space Layout Randomization is a protection technique which randomizes the location where system executables like stack, heap etc are stored in the memory. The attacker should be able to guess the addresses to successfully perform an attack. But this technique makes it harder for the attacker to know its exact location. If the attacker targets an incorrect location, the program will crash, and the system will be alerted about the attempted attack.

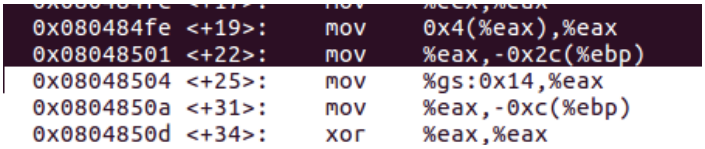
PART B

1. -fno-stack-protector disables the stack protection. When stack protection is ON, a set of data will be added to the attack and while returning the data will be checked to see if an attack has taken place. This is called the Stack Guard. So, by setting the -fno-stack-protector flag, we are disabling the Stack Guard.

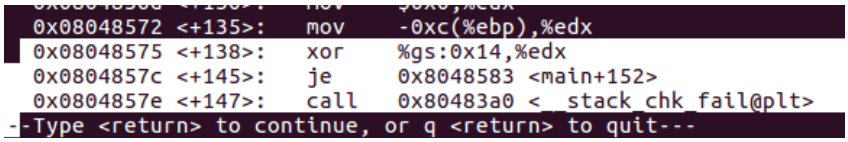
-z execstack makes the stack executable. This is the option where the compiler makes a binary executable or non-executable. The gcc compiler marks the stack as non-executable by default. But by using -z execstack, we can override it and make the stack executable.

In the binary created with these options, the stack canary value register gs is not present as there is no stack protection and the __stack_chk_fail() function is not present.

In the binary created without these options, stack protection is done with the help of stack canary (gs register) and the __stack_chk_fail() is present.



The statements highlighted in white is where the stack canary is stored in the gs register.

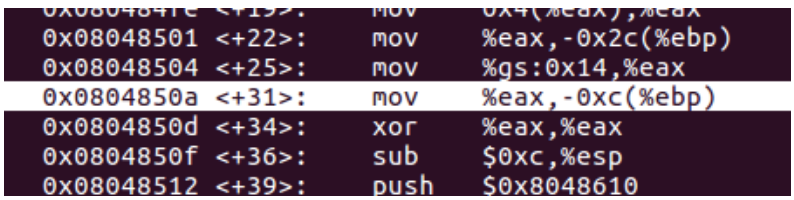


These statements highlighted In white is where the value in the gs register is checked after the program has finished executing and is returning to its caller.
The __stack_chk_fail() is the function which will abort the function that called invoked it as soon as an overflow has been detected.

2. The binary vuln-nossp-exec which is compiled with options -fno-stack-protector and -z execstack is the one that can be exploited by smashing the stack and overflowing a buffer. This is because there the stack protection is turned off and the stack is made executable.

3. After creating the binary using the command : gcc -g -O0 -z execstack -o vuln-ssp-exec vuln.c , I used gdb to debug it. I followed these steps:

   I. Opened binary using the command gdb ./ vuln-ssp-exec
   II. I ran the command disass main to get the dump of assembler code for main function. I did this to obtain the address where the stack canary will be stored.
   III.



As we can see in the above figure, the %gs:0x14 is the stack canary value which is present in the eax register. So, the address at which we need to place the break point is 0x0804850a.

IV. Next, I placed the breakpoint at 0x0804850a using the command br *0x0804850a.
V. Then we run using the command r.
VI. To see the content of the eax, we use the command info r eax. This is the stack canary value.

The canary value changes across multiple compilations/ executions. This is what makes it difficult for the attackers to guess the value.

I tried the above steps for four different binaries and the values are different for every binary. Below are the screenshots.

a. The stack canary in the first attempt is -1536284416.

```
(gdb) br *0x0804850a
Breakpoint 1 at 0x804850a: file vuln.c, line 7.
(gdb) r
Starting program: /home/project1/Desktop/Project_1/Stack_protection/vuln-ssp-exe
c

Breakpoint 1, 0x0804850a in main (argc=1, argv=0xbffff064) at vuln.c:7
7       {
(gdb) info r eax
eax            0xa46e2900       -1536284416
```

b. The stack canary in the second attempt is -1994033920.

```
(gdb) br *0x0804850a
Breakpoint 1 at 0x804850a: file vuln.c, line 7.
(gdb) r
Starting program: /home/project1/Desktop/Project_1/Stack_protection/vuln-ssp-exe
c

Breakpoint 1, 0x0804850a in main (argc=1, argv=0xbffff064) at vuln.c:7
7       {
(gdb) info r eax
eax            0x89257500       -1994033920
```

c. The stack canary in the third attempt is 517197312.

```
(gdb) br *0x0804850a
Breakpoint 1 at 0x804850a: file vuln.c, line 7.
(gdb) r
Starting program: /home/project1/Desktop/Project_1/Stack_protection/vuln-ssp-exe
c

Breakpoint 1, 0x0804850a in main (argc=1, argv=0xbffff064) at vuln.c:7
7       {
(gdb) info r eax
eax            0x1ed3ce00       517197312
```

d. The stack canary in the fourth attempt is 429185536.

```
(gdb) br *0x0804850a
Breakpoint 1 at 0x804850a: file vuln.c, line 7.
(gdb) r
Starting program: /home/project1/Desktop/Project_1/Stack_protection/vuln-ssp-exe
c

Breakpoint 1, 0x0804850a in main (argc=1, argv=0xbffff064) at vuln.c:7
7       {
(gdb) info r eax
eax            0x1994da00       429185536
```

REFERENCES

1. mprotect() - https://docs.oracle.com/cd/E36784_01/html/E36872/mprotect-2.html
2. mprotect() - https://www.ibm.com/support/knowledgecenter/en/ssw_aix_71/m_bostechref/mprotect.html#mprotect__a108c1f10
3. -fno-stack-protector - https://www.keil.com/support/man/docs/armclang_ref/armclang_ref_cjh1548250046139.htm#:~:text=%2Dfno%2Dstack%2Dprotector%20disables,array%20larger%20than%208%20bytes.
4. Basics of Stack Smashing - https://www.techrepublic.com/blog/it-security/basics-of-stack-smashing-attacks-and-defenses-against-them/