Computer Security lab 2018 (Bufferoverflow lab)

Sat Dec 8 11:19:50 +0545 2018

Buffer overflows exploits occur when a program s to write into a buffer beyond the buffers size and get arbitrary code to execute. This can lead to bypassing security protocols, executing parts of code that aren't meant to be executed (changing the flow of control), or gaining control of a machine.

Setup

To disable address randomization in kernel, for ease of bufferoverflow test

```
# to rollback set value to 2
sudo sysctl -w kernel.randomize_va_space=0
```

Also **gcc** compiler implements a security mechanism called `stack guard` to disable it you can use a flag `-fno-stack-protector` during compilation

Tools

objdump

This is a simple tool that will dump an object files information. This will parse the object file and give information on mapped memory for functions, symbols, header information, etc.

Tasks

2.1 man objdump

Answers

- `-x or --all-headers` (Display all available header information, including the symbol table and relocation entries. Using -x is equivalent to specifying all of -a -f -h -p -r -t.)

- `-t or --syms` (Print the symbol table entries of the file.)
- `-M intel or --disassembler-options=intel` (where M is the flag and `intel` is argument)

Sample Example and answers

Sample C program

```
int add_nums(int a, int b){
return a + b;
}

int main(void){
add_nums(17, 25);
}
```

- Answers according to my machine
  - bytecode for ret is `c3`
  - memory location of main function is `000000000000112d`
  - memory location of add$_{num}$ func is `0000000000001119`
  - `push` and `mov` are the two assembly instruction

2.2 GDB

GDB is a debugging tool that allows you to run the compiled file and step through the assembly. Before we look at the simple.c file with gdb, here is a table of common commands.

Answers

- `0x0000555555555140` memory address after the call to add$_{nums}$
-

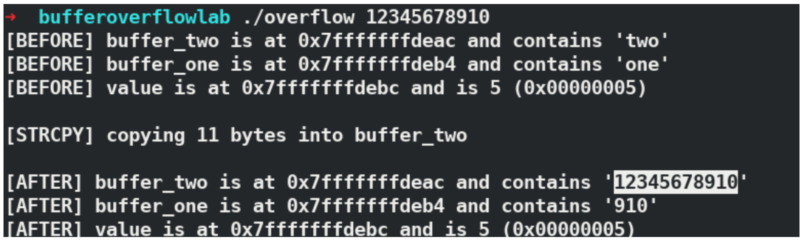
3.1 Simple Buffer Overflow

Answer

- $\text{buffer}_{\text{one}}$=one, $\text{buffer}_{\text{two}}$=two, value=5 and after $\text{buffer}_{\text{one}}$=one, $\text{buffer}_{\text{two}}$=1234, value=5

- I used "12345678910"

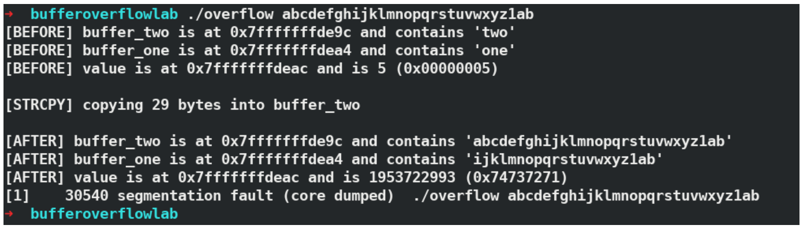
- $\text{buffer}_{\text{two}}$=12345678910 value and $\text{buffer}_{\text{one}}$=910 (my `buffer_two` and `buffer_one` was 8 byte big)

  ```
  ➜  bufferoverflowlab ./overflow 12345678910
  [BEFORE] buffer_two is at 0x7fffffffdeac and contains 'two'
  [BEFORE] buffer_one is at 0x7fffffffdeb4 and contains 'one'
  [BEFORE] value is at 0x7fffffffdebc and is 5 (0x00000005)

  [STRCPY] copying 11 bytes into buffer_two

  [AFTER] buffer_two is at 0x7fffffffdeac and contains '12345678910'
  [AFTER] buffer_one is at 0x7fffffffdeb4 and contains '910'
  [AFTER] value is at 0x7fffffffdebc and is 5 (0x00000005)
  ```

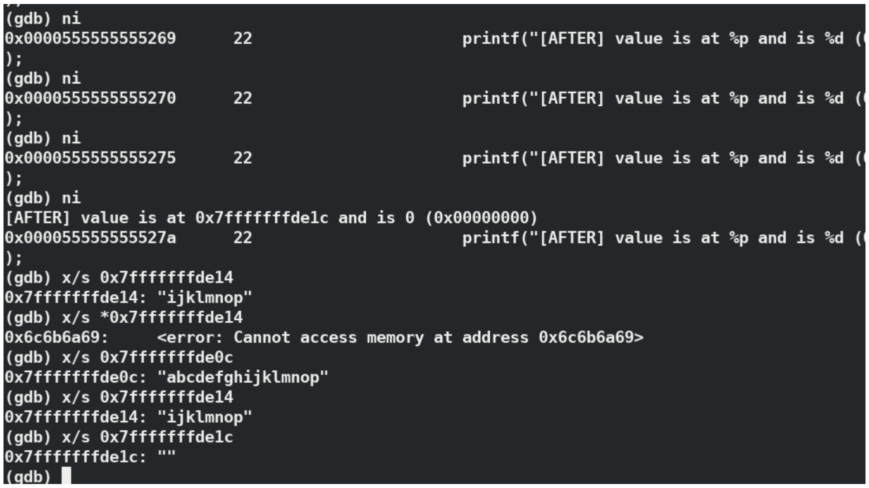
  p

- I got this error after I kept long value: `Program received signal SIGSEGV, Segmentation fault.  0x0000003432343332 in ??  ()`

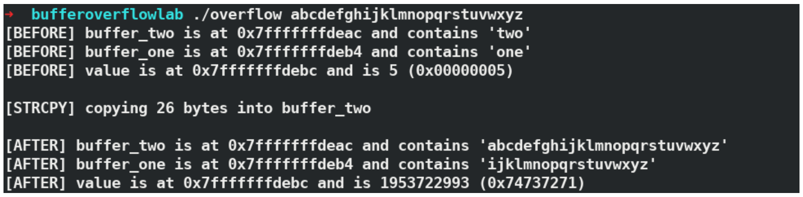
  ```
  ➜  bufferoverflowlab ./overflow abcdefghijklmnopqrstuvwxyz1ab
  [BEFORE] buffer_two is at 0x7fffffffde9c and contains 'two'
  [BEFORE] buffer_one is at 0x7fffffffdea4 and contains 'one'
  [BEFORE] value is at 0x7fffffffdeac and is 5 (0x00000005)

  [STRCPY] copying 29 bytes into buffer_two

  [AFTER] buffer_two is at 0x7fffffffde9c and contains 'abcdefghijklmnopqrstuvwxyz1ab'
  [AFTER] buffer_one is at 0x7fffffffdea4 and contains 'ijklmnopqrstuvwxyz1ab'
  [AFTER] value is at 0x7fffffffdeac and is 1953722993 (0x74737271)
  [1]    30540 segmentation fault (core dumped)  ./overflow abcdefghijklmnopqrstuvwxyz1ab
  ➜  bufferoverflowlab
  ```

- previously $\text{buffer}_{\text{one}}$"one", $\text{buffer}_{\text{two}}$="two", value=5, after $\text{buffer}_{\text{one}}$="ijklmnop", $\text{buffer}_{\text{two}}$="abcdefghijklmnop", value=""(empty)



```
(gdb) ni
0x0000555555555269      22                      printf("[AFTER] value is at %p and is %d (
);
(gdb) ni
0x0000555555555270      22                      printf("[AFTER] value is at %p and is %d (
);
(gdb) ni
0x0000555555555275      22                      printf("[AFTER] value is at %p and is %d (
);
(gdb) ni
[AFTER] value is at 0x7fffffffde1c and is 0 (0x00000000)
0x000055555555527a      22                      printf("[AFTER] value is at %p and is %d (
);
(gdb) x/s 0x7fffffffde14
0x7fffffffde14: "ijklmnop"
(gdb) x/s *0x7fffffffde14
0x6c6b6a69:     <error: Cannot access memory at address 0x6c6b6a69>
(gdb) x/s 0x7fffffffde0c
0x7fffffffde0c: "abcdefghijklmnop"
(gdb) x/s 0x7fffffffde14
0x7fffffffde14: "ijklmnop"
(gdb) x/s 0x7fffffffde1c
0x7fffffffde1c: ""
(gdb) 
```

- the following output was seen by running with fullalphabet `./overflow 'abcdefghijklmnopqrstuvwxyz'`



```
➜  bufferoverflowlab ./overflow abcdefghijklmnopqrstuvwxyz
[BEFORE] buffer_two is at 0x7fffffffdeac and contains 'two'
[BEFORE] buffer_one is at 0x7fffffffdeb4 and contains 'one'
[BEFORE] value is at 0x7fffffffdebc and is 5 (0x00000005)

[STRCPY] copying 26 bytes into buffer_two

[AFTER] buffer_two is at 0x7fffffffdeac and contains 'abcdefghijklmnopqrstuvwxyz'
[AFTER] buffer_one is at 0x7fffffffdeb4 and contains 'ijklmnopqrstuvwxyz'
[AFTER] value is at 0x7fffffffdebc and is 1953722993 (0x74737271)
```

- Significance about above command is that, whenever the input is given more than 8 bytes it overflows towards next buffer regardless of any change in memory addresses.