

CS6570 Secure Systems Engineering

Lab 1

Akshith Sriram CS19B005, Bhanu Shashank CS19B043

Q1. Draw the different snapshots of the content of the stackframe along with different values of registers like ebp, esp etc for each step of the attack.

Stack state before making a function call to do_stuff

```

Reading symbols from ./vegas1...done.
(gdb) b 61
Breakpoint 1 at 0x8048a03: file vegas1.c, line 61.
(gdb) r
Starting program: /home/nptel/assignments/vegas1
Welcome to my guessing game!

Breakpoint 1, main (argc=1, argv=0xffffd0e4) at vegas1.c:61
61          res = do_stuff();
(gdb) x/16x $esp
0xffffcfff: 0x080eb00c    0x00000008e    0x00000000    0x0000003e9
0xffffd000: 0x080eb070    0xffffd020    0x00000000    0x08048c51
0xffffd010: 0x080eb00c    0x00000008e    0x00000000    0x08048c51
0xffffd020: 0x00000001    0xffffd0e4    0xffffd0ec    0xffffd044

```

	Stack before execution of fgets(guess, 100, stdin)			Stack after execution of fgets (buffer overflow)	
return address to main	0x08048a08	0xffffcec	return address to main	0x08048a08	0xffffcec
ebp of previous stack frame	0x080b000a	0xffffce0	ebp of previous stack frame	0x080b000a	0xffffce0
ans (local variable) generated by rand()	0x00000054	0xffffcdc	ans (local variable)	0x00000061	0xffffcdc
g (local variable)	0xffffd008	0xffffcd8	g (local variable)	0x32323232	0xffffcd8
random[2..5]	0x0000008e	0xffffcd4	random[2..5]	0x31313131	0xffffcd4
random[0..1]	0x080e	0xffffcd2	random[0..1]	0x3131	0xffffcd2
guess[14..15]	0xb00c	0xffffcfd0	guess[14..15]	0x3131	0xffffcfd0
guess[10..13]	0x080481a8	0xffffcfc	guess[10..13]	0x31313131	0xffffcfc
guess[6..9]	0x00000000	0xffffcfc8	guess[6..9]	0x31313131	0xffffcfc8
guess[2..5]	0x080ec024	0xffffcfc4	guess[2..5]	0x31313100	0xffffcfc4
guess[0..1]	0x080d	0xffffcfc2	guess[0..1]	0x3739	0xffffcfc2
esp →	0x637c	0xffffcfc0	esp →	0x637c	0xffffcfc0

Final Stack state after execution of atol (winning the game)		
return address to main	0x08048a08	0xffffcec
ebp of previous stack frame	0x080b000a	0xffffce0
ans (local variable)	0x00000061	0xffffcdc
g (local variable)	0x00000061	0xffffcd8
random[2..5]	0x31313131	0xffffcd4
random[0..1]	0x3131	0xffffcd2
guess[14..15]	0x3131	0xffffcfd0
guess[10..13]	0x31313131	0xffffcfc8
guess[6..9]	0x31313131	0xffffcfc4
guess[2..5]	0x31313100	0xffffcfc2
guess[0..1]	0x3739	0xffffcfc0
esp →	0x637c	0xffffcfc0

Q2. List all the bugs/vulnerabilities present in the program.

There are two major bugs in the program:

1. `fgets(guess, 100, stdin)` in `do_stuff` function.
2. `fgets(winner, 360, stdin)` in `win` function.

The above are vulnerabilities because of the incorrect/careless usage of the otherwise safe `fgets` function. The maximum limit on the number of characters to read (second argument in `fgets`) in each case is larger than the actual size of the buffers `guess` (16 bytes) and `winner` (100 bytes). This can potentially cause a buffer overflow.

Q3. If there are vulnerabilities, then which of them can be exploited to subvert execution?

Both of the above-mentioned vulnerabilities may be exploited to subvert execution. We can easily give a large input string and modify the memory location where the return address is stored. We can also modify values stored inside some crucial local variables to alter the way the program usually behaves.

Q4. As an attacker, how can you create an exploit that cheats to 'win' the game? Use one of the vulnerabilities that you found.

To 'win' the game, we have to pass the condition `if(g == ans)`. All we have to do is make `g` equal to `ans`. It is not possible for us to accurately 'guess' what the random number stored in `g` is. Instead what we can do is take advantage of the vulnerability in the `do_stuff` function in the form of `fgets`.

Based on the organization of variables in the stack (as shown in Q1), we can change the contents of the memory location `ans` by overflowing the `guess` buffer. We have to store values inside `guess` and `ans` such that `atoi(guess) == ans`. We chose to store the value `97` in both locations.

`atoi` function converts the initial portion of the character array (till the null termination) to long type and returns that number.

If `guess = {'9', '7', '\0',}`, then `g = atoi(guess) = 97`

From the diagram shown in Q1, the variable `ans` is located at an offset of 26 bytes from base address of buffer `guess`. So, we can fill the first few bytes of `guess` with `'9', '7', '\0'` and fill the remaining 23 characters with any random hexadecimal values. The next 4 bytes (size of the long variable `ans`) should then be filled with `'a', '\0', '\0', '\0'` (following the little-endian system) so that `g == ans` is satisfied. Note that the ASCII value of `'a'` is 97, and the same will be stored in `ans` variable.

One possible exploit string is `"97\011111111111111111112222a\0\0\0"`, where `\0` is the null termination character.