Computer Security

05r. Assignment 4 discussion

Paul Krzyzanowski
Rutgers University
Spring 2017
Assignment 4 hints

Level 0 Goal:
– Overflow the buffer to change the return address on the stack
– When the function `getbuf` returns, make it go to `smoke`

First, we need to find the address of the `smoke` function
– Two ways to do this:
  1. Use the `nm` (display name list) command to dump the symbol table
     $ nm bufbomb | grep smoke
  1. Use `gdb` and print the value of `smoke`
     $ gdb bufbomb
     (gdb) print smoke
Assignment 4 hints: level 0

Now create the exploit string:
– Fill the 12 bytes of the buffer
– Fill 4 more bytes to overwrite the saved %ebp register (frame pointer)
– Write the return address to overwrite the saved return address
– Create a file (e.g., exploit-0.txt) with contents:

```
00 11 22 33 44 55 66 77 88 99 aa bb 1a 1b 1c 1d RR RR RR RR
```

This could be anything – just fill the buffer – but let’s pick something we can easily recognize in gdb

Run it

```
$ cat exploit-0.txt | ./sendstring > exploit-0
$ bufbomb < exploit-0
```
Intel uses little endian encodings

The address 0x12345678

Will be written to the buffer as

0x78 0x56 0x34 0x12

Make sure it’s in the right order in your buffer.
What if it doesn’t work?

You’ll have to debug

$ gdb bufbomb  
  \textit{start the debugger}

(gdb) break getbuf  
  \textit{set a breakpoint at getbuf}

(gdb) run -t your_net_id < exploit-0
  \textit{run the program to the breakpoint}

Breakpoint 1, 0x08048aa8 in getbuf ()
What if it doesn’t work?

(gdb) disas - disassemble the current function
  0x08048aa2 <+0>: push %ebp
  0x08048aa3 <+1>: mov %esp,%ebp
  0x08048aa5 <+3>: sub $0x10,%esp
=> 0x08048aa8 <+6>: lea -0xc(%ebp),%eax
  0x08048aab <+9>: mov %eax,(%esp)
  0x08048aae <+12>: call 0x8048bf1 <Gets>
  0x08048ab3 <+17>: mov $0x1,%eax
  0x08048ab8 <+22>: leave
  0x08048ab9 <+23>: ret
End of assembler dump.
(gdb) break *0x08048ab3 - set a break after call Gets
(gdb) c - run to the next breakpoint
(gdb) x/20b $sp - print 20 bytes at the stack pointer
    (buf starts after the first four bytes)

See if the data in the buffer is what you expected
Levels 1 hint

• You will need to give \textit{fizz} a parameter

• This means that you will need to add extra data after the address of \textit{fizz} to modify what’s on the stack when \textit{getbuf} returns

• But think carefully about what the stack should look like
Level 2 hint

• You will need to write code to set `global_int` to `cookie`

• You can easily find the value of `global_int` via `gdb`

• But you also need to find the start of the buffer (`buf`)

• You can find this by looking at the stack pointer in `getbuf` and figure out where `getbuf` allocates `buf` (look at the disassembly)

  … or set a breakpoint in `Gets` and look around there
Level 2 hint

• To set the buffer, you’ll need to write a few lines of assembly code

• If you don’t know it, you can figure it out
  – Write a small C function that simply sets a global into to the value
  – Compile it with `cc -S t.c`
  – That creates an assembler file `t.s`
  – Look through it. You’ll see the instruction that sets a value. You’ll also see how you can push something on the stack and how you can return

• The exploit code will go at the start of your buffer
  – So the return address that you overwrite will have to be an address to the start of the buffer
The end