Memory + Assembly Language: The Security Arms Race

- Learning Objectives
  - Explain how understanding how programs use memory and being able to read assembly creates opportunities to wreak havoc.
  - Explain how compilers (and possibly other tools) help protect you/programs from nefarious behavior.
  - Avoid using unsafe constructs in your programs.

When Tech Goes Mainstream

9/23/15: Google Hacker Digs Up More Kaspersky Zero Days, Demands Better from Anti-Virus Industry … "Ormandy found the vendor hadn't turned on security mechanisms designed to Prevent certain buffer overflows."

9/22/15: Adobe releases surprise security update; 23 critical vulnerabilities fixed … "The security flaws fixed in this update, all deemed critical, including a type confusion vulnerability, use-after-free flaws, buffer overflows, and memory corruption vulnerabilities which could lead to remote code execution."

9/17/15: Cisco addresses vulnerabilities in several products … "a third Wednesday advisor explained that vulnerable version of Cisco's TelePresence Server contain a buffer overflow attack.

- http://www.forbes.com/sites/thomasbrewster/2015/09/23/google-ormandy-finds-kaspersky-0days/
- http://www.zdnet.com/article/adobe-releases-surprise-security-update-23-critical-vulnerabilities-fixed/
- http://www.scmagazine.com/cisco-addresses-vulnerabilities-in-several-products/article/439268/

Recall …

- One of our first in-class exercises showed you that it was relatively easy to overwrite the caller's return address on the stack.
- If you can overwrite the caller's return address with an address of your choosing, then you just might be able to execute code that the program designer never intended for you to execute!

Consider the following program

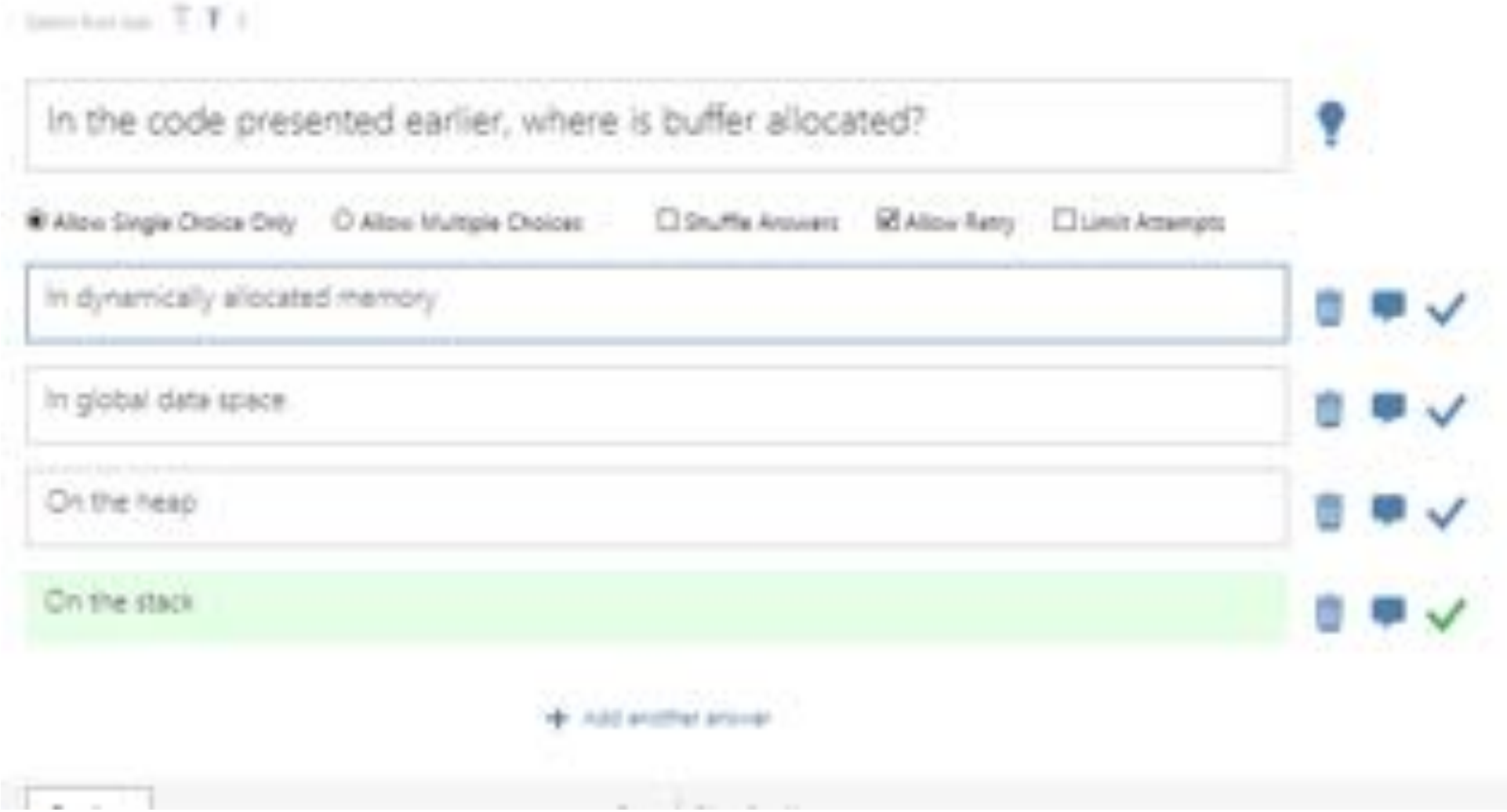
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
     char buffer[100];
     if (gets(buffer))
         printf("Read %d character(s)\n", strlen(buffer));
     else
         printf("Read nothing\n");
 }
```

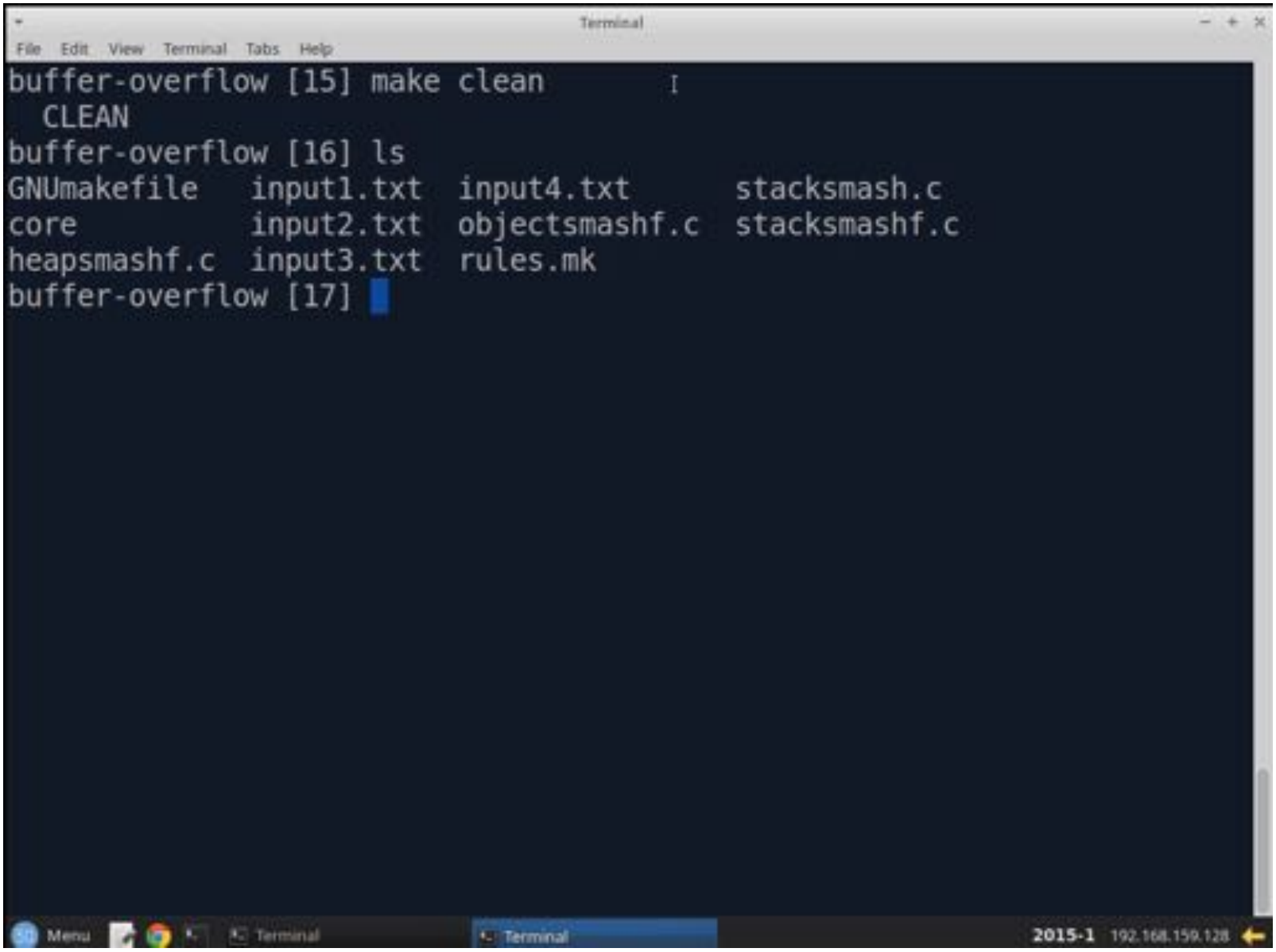
cs61-videos/buffer-overflow/stacksmash.c

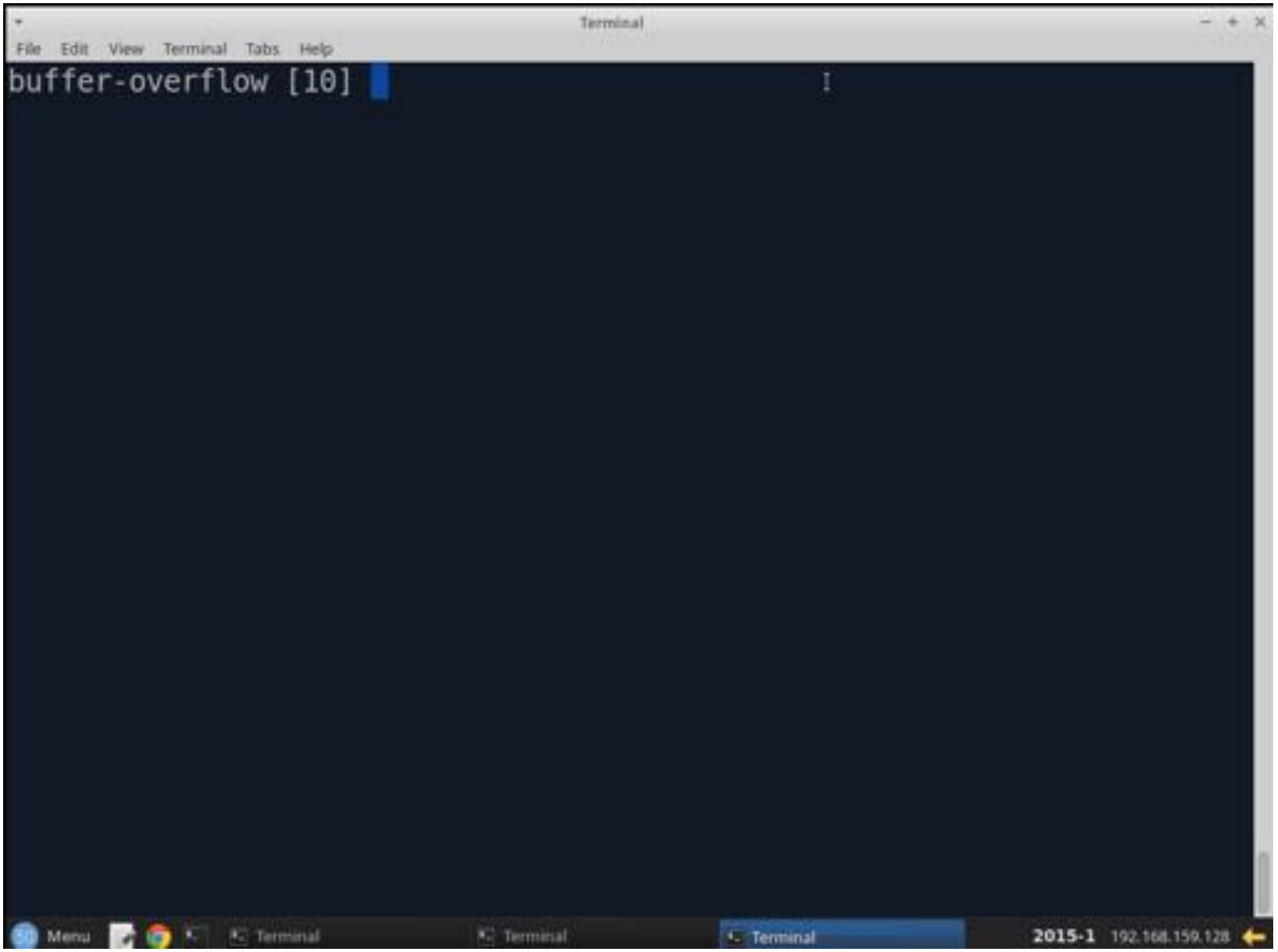

What problem(s) did you identify in the code?
What might happen if I entered a very long string?

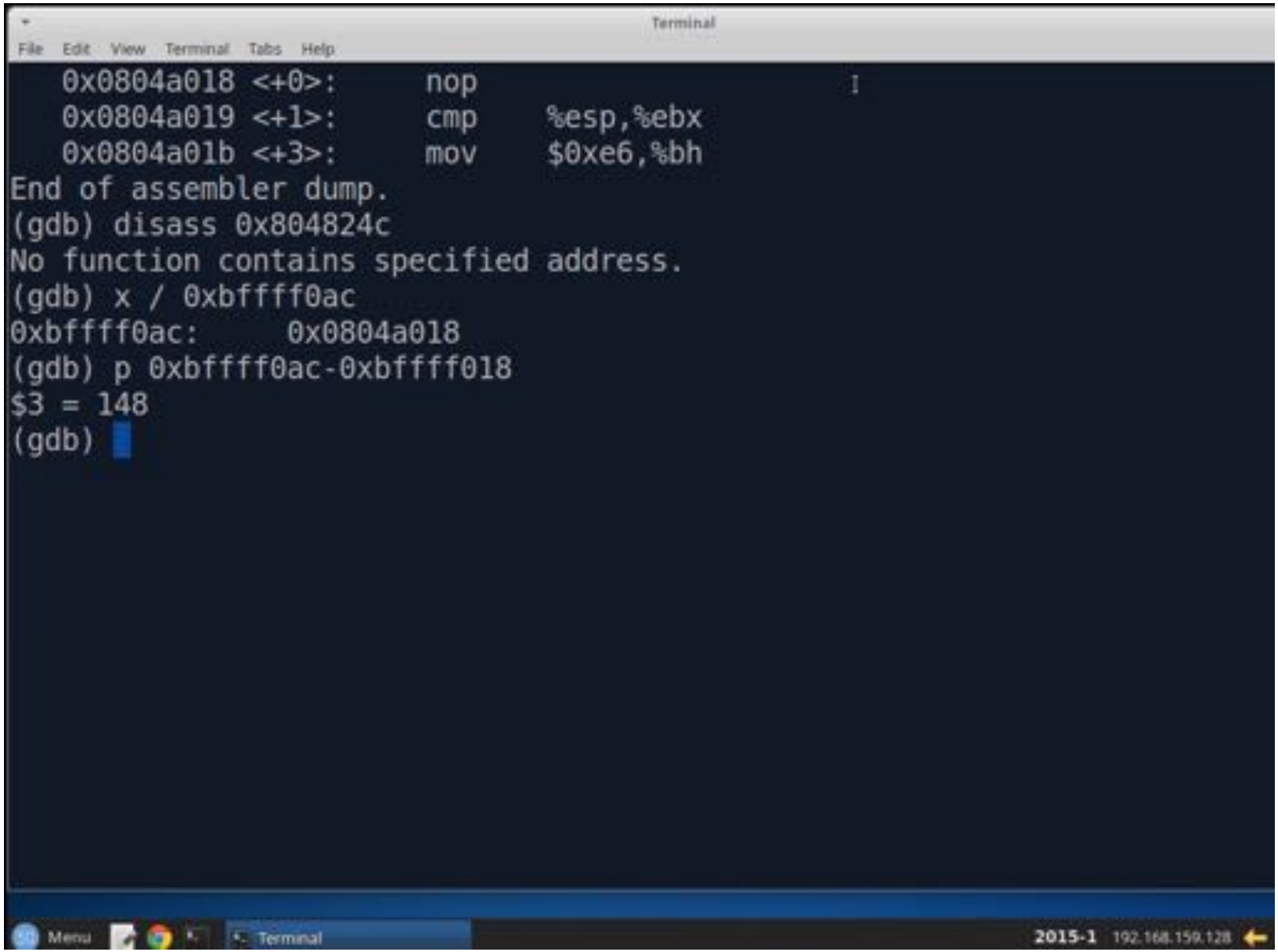
`gets` is evil

- Check out the manpage …
- It starts out innocently enough, "`gets()` reads a line from `stdin` into the buffer pointed to by `s` until either a terminating newline or EOF, which it replaces with a null byte ('\0').
- Then it gets interesting, "*No check for buffer overrun is performed (see BUGS below).*"
- Then it gets downright fun, "Never use `gets()`. Because it is impossible to tell without knowing the data in advance how many characters `gets()` will read, and because `gets()` will continue to store characters past the end of the buffer, it is extremely dangerous to use. It has been used to break computer security. Use `fgets()` instead."


In the code presented earlier, where is buffer allocated?
Allow Single Choice Only
Allow Multiple Choices
Shuffle Answers
Allow Retry
Limit Attempts
In dynamically allocated memory
In global data space
On the heap
On the stack

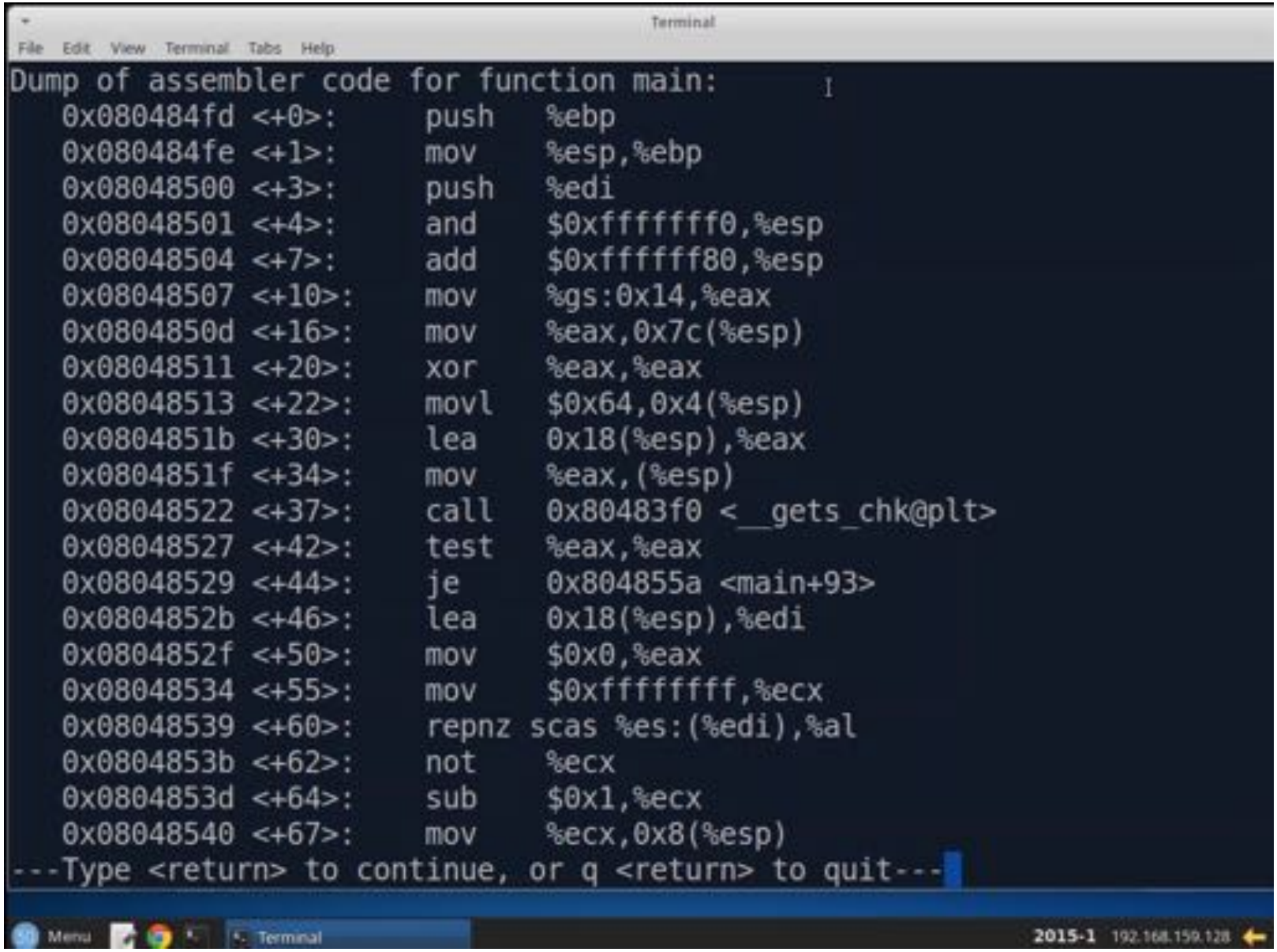

Terminal
File Edit View Terminal Tabs Help
buffer-overflow [15] make clean
CLEAN
buffer-overflow [16] ls
GNUmakefile input1.txt input4.txt stacksmash.c
core input2.txt objectsmashf.c stacksmashf.c
heapsmashf.c input3.txt rules.mk
buffer-overflow [17]
Menu
Terminal
Terminal
2015-1
192.168.159.128


Terminal
File Edit View Terminal Tabs Help
buffer-overflow [10]
Menu
Terminal
Terminal
Terminal
2015-1
192.168.159.128


Terminal
File Edit View Terminal Tabs Help
0x0804a018 <+0>: nop
0x0804a019 <+1>: cmp %esp,%ebx
0x0804a01b <+3>: mov $0xe6,%bh
End of assembler dump.
(gdb) disass 0x804824c
No function contains specified address.
(gdb) x / 0xbffff0ac
0xbffff0ac: 0x0804a018
(gdb) p 0xbffff0ac-0xbffff018
$3 = 148
(gdb)
Menu
Terminal
2015-1

A Sneakier Version (stacksmashf)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int read_line(char* buffer) {
    if (gets(buffer))
        return 1;
    else
        return 0;
}
int main() {
    char buffer[100];
    if (read_line(buffer))
        printf("Read %d character(s)\n", strlen(buffer));
    else
        printf("Read nothing\n");
}
```


Terminal
File Edit View Terminal Tabs Help
Dump of assembler code for function main:
0x080484fd <+0>: push %ebp
0x080484fe <+1>: mov %esp,%ebp
0x08048500 <+3>: push %edi
0x08048501 <+4>: and $0xfffffff0,%esp
0x08048504 <+7>: add $0xffffff80,%esp
0x08048507 <+10>: mov %gs:0x14,%eax
0x0804850d <+16>: mov %eax,0x7c(%esp)
0x08048511 <+20>: xor %eax,%eax
0x08048513 <+22>: movl $0x64,0x4(%esp)
0x0804851b <+30>: lea 0x18(%esp),%eax
0x0804851f <+34>: mov %eax,(%esp)
0x08048522 <+37>: call 0x80483f0 <__gets_chk@plt>
0x08048527 <+42>: test %eax,%eax
0x08048529 <+44>: je 0x804855a <main+93>
0x0804852b <+46>: lea 0x18(%esp),%edi
0x0804852f <+50>: mov $0x0,%eax
0x08048534 <+55>: mov $0xffffffff,%ecx
0x08048539 <+60>: repnz scas %es:(%edi),%al
0x0804853b <+62>: not %ecx
0x0804853d <+64>: sub $0x1,%ecx
0x08048540 <+67>: mov %ecx,0x8(%esp)
---Type <return> to continue, or q <return> to quit---
Menu
Terminal
2015-1 192.168.159.128

Wrapping Up

- Understanding memory layout and being able to read assembly gives you great power – use it wisely.
  - Safely
  - Productively
  - Constructively

- Be a white hat!