PURDUE UNIVERSITY®

CS 240: Programming in C

Lecture 24: Buffer Overflows Core Files, goto, and Makefiles

Prof. Jeff Turkstra

Announcements

- Final Exam is Thursday, May 8
  - 10:30am – 12:30pm
  - Short answer and coding questions
  - Some multiple choice and True/False
  - Accommodated exams overlap that time
    - Email from Megan in the next week or so

Homework 13

Address Space Layout Randomization

- ASLR changes the addresses of many things every time a program executes
  - Harder to exploit certain vulnerabilities

```
$ setarch x86_64 -R my_binary
```

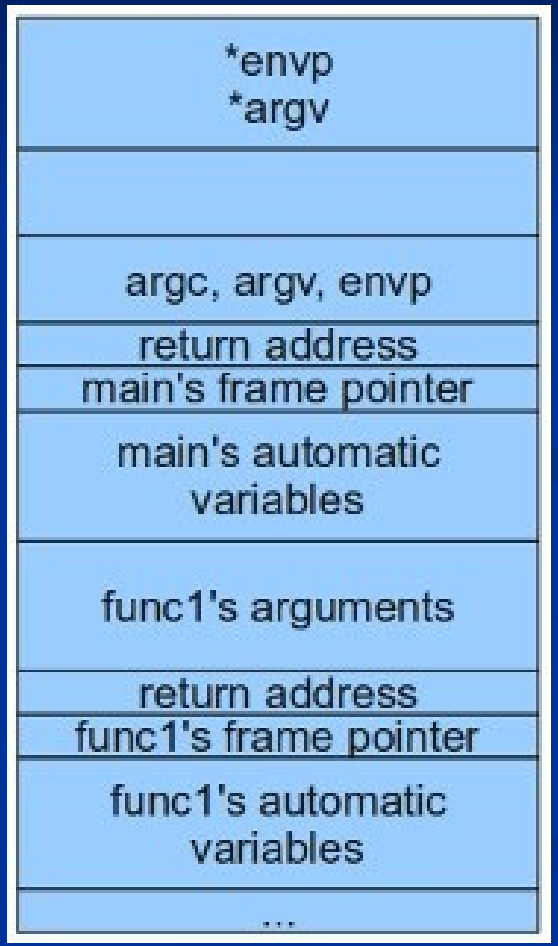
```
$ gcc -fno-stack-protector -z
execstack -o doit doit.c
```

- When run, we will get a dump of 112 bytes of memory starting at the address of local
- We know basically what the output should look like, right?

output...

```
Address of main is 0x401285
Address of local is 0x7ffd4c42d1dc
0x7ffd4c42d1e0: e0 12 40 00 00 00 00 00 ??@?????
0x7ffd4c42d1d8: 00 00 00 00 ef be ef be ????????
0x7ffd4c42d1d0: c0 d2 42 4c fd 7f 00 00 ??BL????
0x7ffd4c42d1c8: 60 10 40 00 01 00 00 00 `?@?????
0x7ffd4c42d1c0: c8 d2 42 4c fd 7f 00 00 ??BL????
0x7ffd4c42d1b8: d8 d2 42 4c fd 7f 00 00 ??BL????
0x7ffd4c42d1b0: c0 ae f7 ac a6 7f 00 00 ????????
0x7ffd4c42d1a8: d3 12 40 00 00 00 00 00 ??@?????
0x7ffd4c42d1a0: e0 d1 42 4c fd 7f 00 00 ??BL????
0x7ffd4c42d198: 50 71 f9 ac ad fb ca de Pq??????
0x7ffd4c42d190: 00 00 00 00 00 00 00 00 ????????
0x7ffd4c42d188: b6 d1 42 4c aa aa aa aa ??BL????
0x7ffd4c42d180: c2 00 00 00 00 00 00 00 ????????
0x7ffd4c42d178: 82 12 40 00 00 00 00 00 ??@?????
```

Stack layout

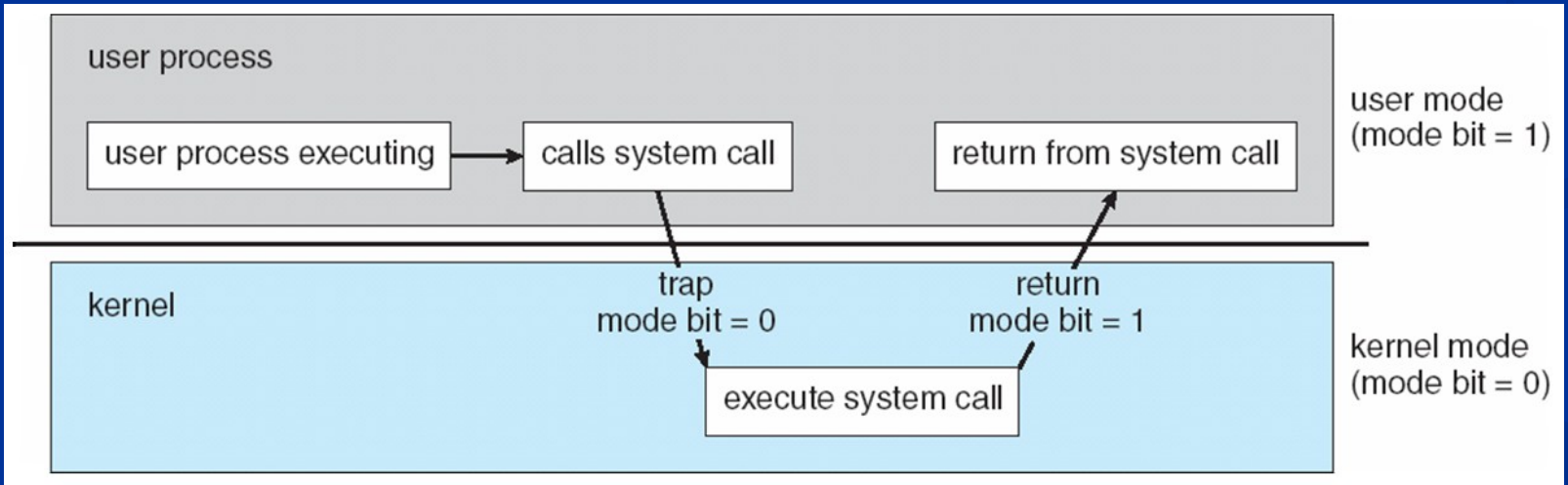


Kernel invocation

- What causes the switch to kernel mode?
  - System calls
  - Page faults
  - Signals
  - Hardware

System calls

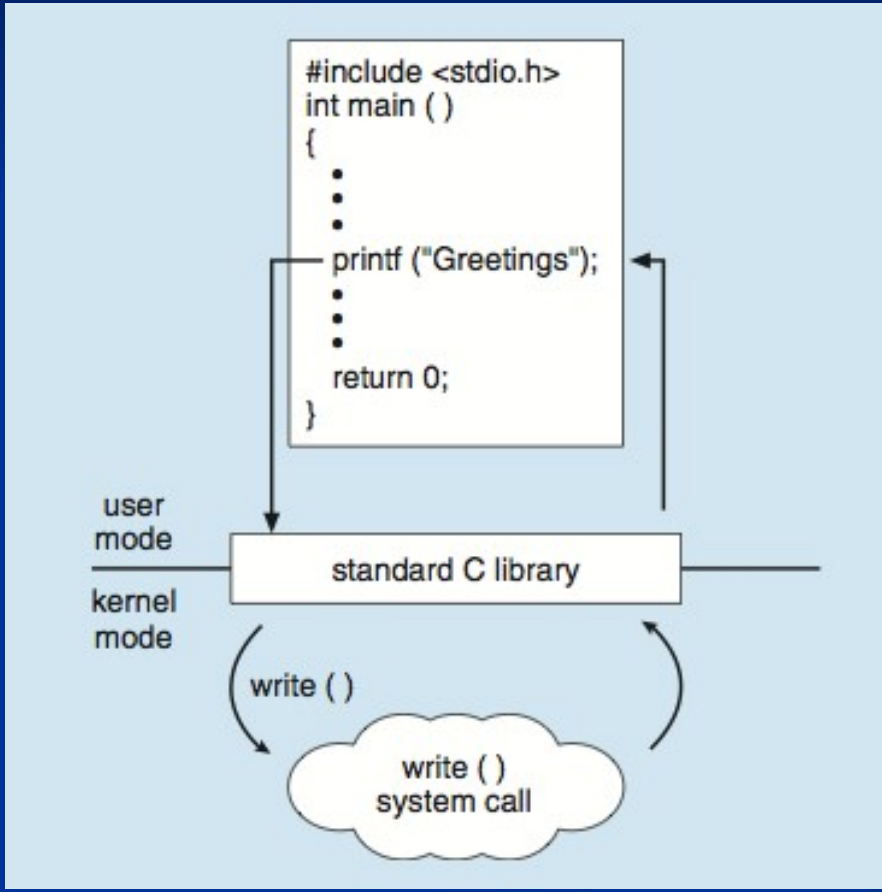
- System calls are the interface between processes and the OS kernel



System call types

- Process management
  - Create, terminate, execute, wait, etc
- File management
  - Create file, delete, open, close, read write, getattr, setattr, etc
- Device management
  - ioctl, read, write, etc
- Information management
  - getpid, alarm, sleep, etc
- Communication – between processes
  - pipe, shmget, mmap, etc

Standard C library



open() system call

```
int open(const char *pathname, int flags[, mode_t mode]);
```

- Flags includes:
  - Access mode (O_RDONLY, O_WRONLY, O_RDRWR) – required
  - File creation flags (O_CLOEXEC, O_CREAT, O_TRUNC, etc) – optional
  - File status flags (O_APPEND, O_SYNC, O_NONBLOCK, etc)
- Mode is your usual file creation mode

close() system call

```
int close(int fd);
```

- Decrements the reference count for the appropriate open file object
- Object is reclaimed if reference count == 0
- Returns -1 on error and sets errno
- Failing to close() fds results in a file descriptor leak
  - Arguably worse than a memory leak

...and

- `ssize_t read(int fd, void *buf, size_t count);`
- `ssize_t write(int fd, void *buf, size_t count);`

Using assembly language in C

- C gets compiled into a lower-level language called assembly language where each instruction represents one CPU instruction
- We generally do not have control over which assembly language instructions are chosen, but most compilers support a way of embedding specific instructions

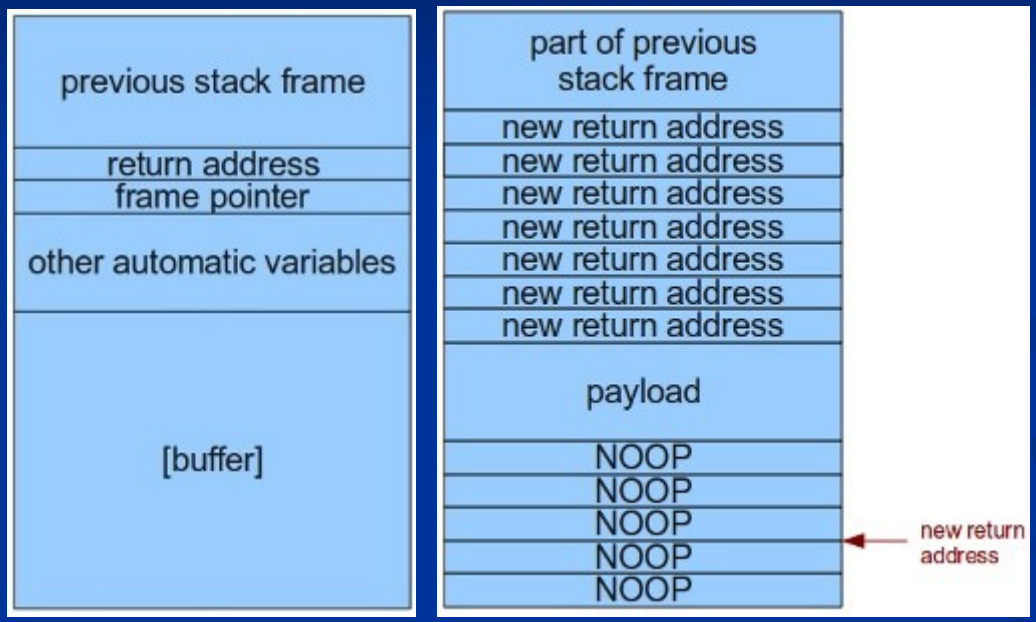
Example of asm in C

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

int main(int argc, char *argv[]) {
  unsigned char x;
  x = atoi(argv[1]);

  __asm__ __volatile__("add $1,%0" : "=r"(x) : "0"(x));

  printf("%d\n", x);
  return x;
}
```

More examples

- Shell
- chmod

A small server


previous stack frame
return address
frame pointer
other automatic variables
[buffer]
part of previous stack frame
new return address
new return address
new return address
new return address
new return address
new return address
new return address
payload
NOOP
NOOP
NOOP
NOOP
NOOP
new return address

Security

- You have a very, very small taste of what kind of problems can arise in terms of program security
- We've only touched the "tip of the iceberg" in terms of buffer overflows
- If you want to know more, check out:
  - https://turkeyland.net/projects/overflow/
  - ...or find "Smashing the Stack for Fun and Profit" using a search engine
- There are many, many other types of vulnerabilities
- If you enjoy this stuff, take a security course!

Takehome Quiz 9

- 1. Given the function and stack dump:

```
void hello(int arg) {
  int local = 0xdecafbad
}
```

```
(1) 0x7ffc90298f88: 48 62 83 b0 4d 7f 00 00 Hb??M???
(2) 0x7ffc90298f80: 20 90 29 90 fc 7f 00 00  ?)?????
(3) 0x7ffc90298f78: 93 12 40 00 00 00 00 00 ??@?????
(4) 0x7ffc90298f70: 80 8f 29 90 fc 7f 00 00 ??)?????
(5) 0x7ffc90298f68: 10 94 a7 b0 ad fb ca de ????????
(6) 0x7ffc90298f60: 00 00 00 00 00 00 00 00 ????????
(7) 0x7ffc90298f58: 00 00 00 00 11 11 11 11 ????????
```

  - a. Which line number contains the local variable?
  - b. Which line number contains the return address?
  - c. Which line number contains the argument, assuming the function is passed the value 0x11111111?
- 2. On x86_64 systems, which instruction is used to initiate a system call?

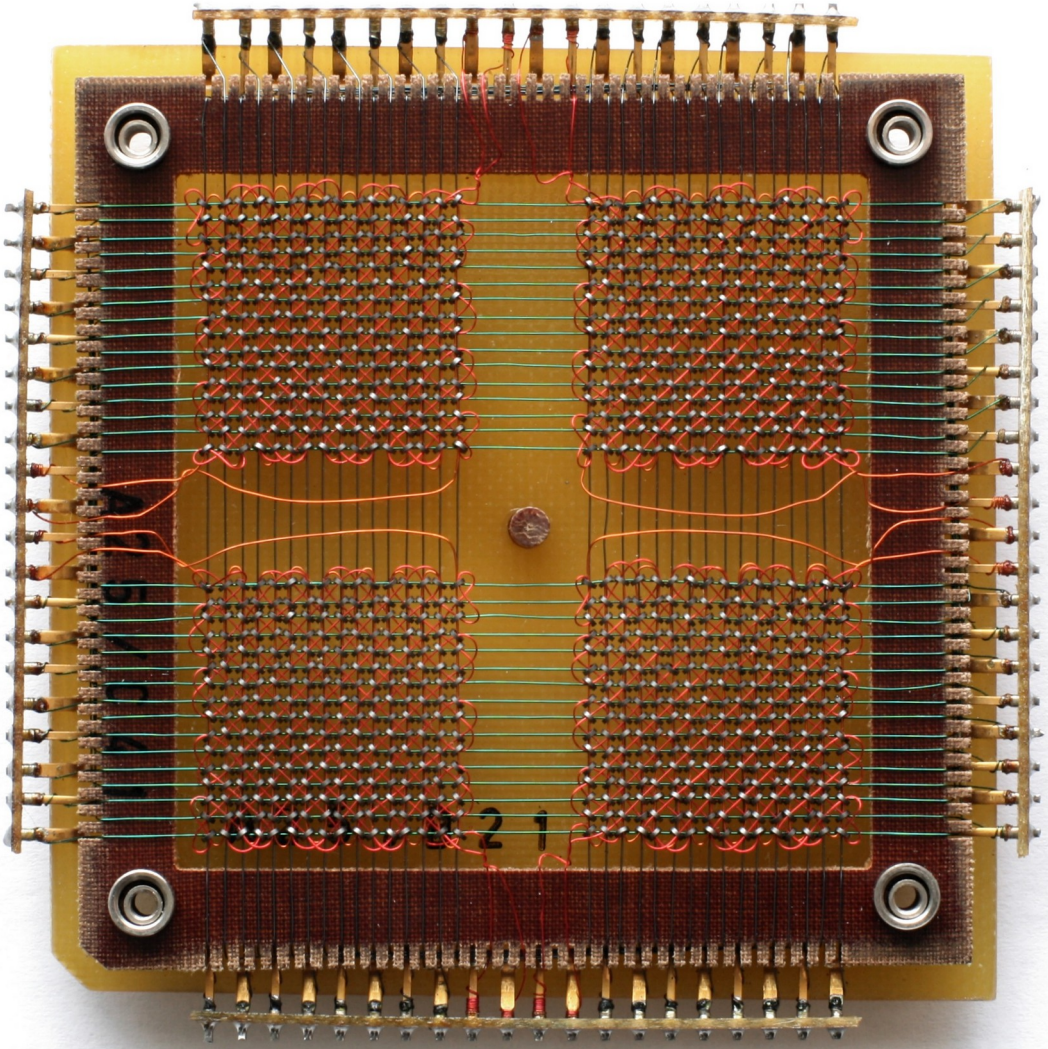
Purdue Trivia

- Old Oaken Bucket is one of the oldest football trophies in the nation
  - Found on a farm in Southern Indiana
  - First competition in 1925 lead to a tie
  - Otherwise add a P or I depending on who wins


FOOTBALL TROPHY

Core Files

- Does anyone know what “core” memory was?
- When your program has an unrecoverable error, the operating system saves the heap/stack memory at the exact time of the failure into a file named “core”.
- You can use the core file with the debugger



Core dump file

- $ man 5 core
- May have to enable it (e.g., on data.cs.purdue.edu)
  - $ ulimit -c unlimited

The Official Disclaimer with respect to “goto”

- "For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of go to statements in the programs they produce. More recently I discovered why the use of the go to statement has such disastrous effects, and I became convinced that the go to statement should be abolished from all "higher level" programming languages (i.e. everything except, perhaps, plain machine code). At that time I did not attach too much importance to this discovery; I now submit my considerations for publication because in very recent discussions in which the subject turned up, I have been urged to do so."
  --Edsger W. Dijkstra, March 1968, Comm. of ACM,
  "Go To Statement Considered Harmful"

Why is goto bad?

- Dijkstra made the case that goto was harmful for the following reasons:
  - It prevents the compiler from being able to make “nice” computer-sciency reductions of the program
  - It makes your code unreadable
  - It is really not necessary
    - You can always rewrite to have the same functionality without goto

Why does C have a goto?

- Because…
  - The compiler doesn't have any more difficulty analyzing a program with gotos in it
  - It often makes the program clearer to read
  - It is very useful – at a certain level, at least
- Contradictions?
- More enlightened languages have even more dangerous control flow operations

What does goto look like?

- You can define labels and goto those labels…

```
int func(int x) {
  int sum = 0;

 again:
  sum = sum + x;
  x = x – 1;
  if (x <= 0)
    goto get_out;
  else
    goto again;
 get_out:
  return sum;
}
```

How can goto make a program clearer to read?

- When you really need to ditch the control flow of your program and take drastic measures:

```
start_over:
 for (int x = 0; x < 5000; x++) {
   ptr = array[x];
   while (ptr->val < level) {
     while (ptr->next != 0 && ptr->val < level) {
       if (ptr->total == 0) {
         level++;
         goto start_over;
       }
     }
     sum += ptr->total;
   }
 }
}
```

When is goto useful?

- When it is necessary to break out of deeply nested loops (previous example)
- When you're building a state machine in software

- In general, you should still avoid using gotos unless there is a really good reason

Makefile

- Simple way to help organize code compilation
- Composed of rules
  - Target – usually a file to generate
    - Can be an action (“make clean”)
  - Prerequisites – used to create the target
  - Recipe – action to carry out
    - Must start with a tab!

```
gcc -o hello hello.c hellofunc.c -I.
```

Simple, hard coded

```
hello: hello.c hellofunc.c
    gcc -o hello hello.c hellofunc.c -I.
```

Or...

```
CC=gcc
CFLAGS=-I.

hello: hello.o hellofunc.o
    $(CC) -o hello hello.o hellofunc.o $(CFLAGS)
```

More generic

```
CC=gcc
CFLAGS=-I.
DEPS = hello.h

%.o: %.c $(DEPS)
    $(CC) -c -o $@ $< $(CFLAGS)

hello: hello.o hellofunc.o
    gcc -o hello hello.o hellofunc.o -I.
```

More Variables

```
CC=gcc
CFLAGS=-I.
DEPS = hellomake.h
OBJ=hello.o hellofunc.o


%.o: %.c $(DEPS)
	$(CC) -c -o $@ $< $(CFLAGS)


hello: $(OBJ)
	gcc -o $@ $^ $(CFLAGS)
```

```
.PHONY: clean

clean:
    rm -f $(ODIR)/*.o *~ core $(INCDIR)/*~
```

Lot's More

- https://www.gnu.org/software/make/manual/html_node/index.html

Boiler Up!