

Welcome to CS 110L



Ryan Eberhardt and Julio Ballista
March 30, 2021

Zoom norms

- Please enable video (if you have one)
- Try to mute yourself when not speaking
- Feel free to just unmute and start talking if you have a question

Who are we?

Julio Ballista

- Junior in CS (2022!) (Systems track)
- Interested in operating systems, systems architecture / design, security
- I took 110L last year where I learned Rust for the first time
- I've been jump roping since Frosh year (Stanford Jump Rope :D)
- Got into rhythm games this pandemic

Ryan Eberhardt

- Coterm focused on systems and security
- I have two cats
- I love doing pottery, photography, and listening to music



HUGE thanks to Will Crichton for course material, advice, and feedback.

Also thanks to Armin Namavari for developing this class last spring!

Who are you?

Who are you?

Fun and quirky community of about 35 students (including auditors)

Who are you?

Why are you taking this class?

- *I've been really interested in learning Rust recently, and hearing that there was a class that helps to teach it this quarter was perfect timing.*
- *I want exposure to Rust and to understand common safety and robustness pitfalls.*
- *I am currently considering the CS systems track, so I think it is pretty much required I learn about safety in systems programming.*
- *I'm interested in getting experience with Rust, and in brushing up on / extending my 110 knowledge since I took it in Spring 2020 when I was still getting used to virtual learning. The projects sound really cool!*

Who are you?

Have you heard anything about Rust before?

- Most people: “*Nope.*”
- Note: If you have prior Rust experience (three people), you will likely have seen most of the content from the first half of the class. (Feel free to stay for the second half!)

Who are you?

Tell us something about you that has nothing to do with this class.

11 responses

I love lego

I have a twin sister!

I love coming up with puns and over the past couple months have really loved making polymer clay jewelry!

I like playing Kerbal Space Program!

One of my hobbies is language acquisition!

I love ballroom dance!

I'm an aviation geek and love to go plane spotting with friends. I just went over break!

I can make really good tomato soup.

I can't sing to save my life.

Say hi on #social!
(Let us know if you need a Slack invite.)

Why are we here?

“Convert a String to Uppercase in C,” taken VERBATIM from [Tutorials Point](#)

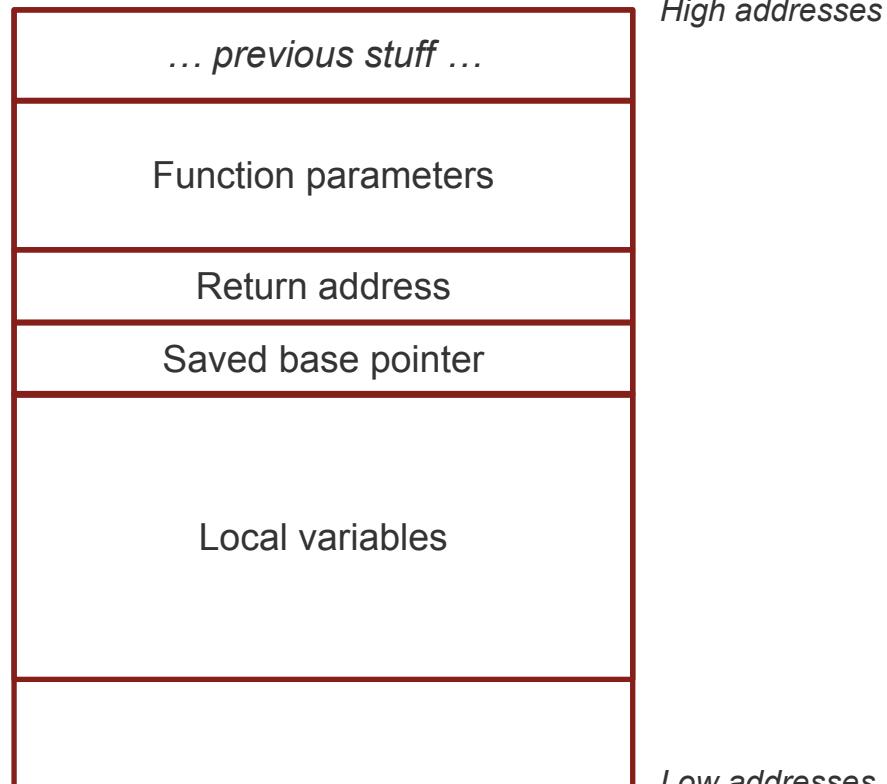
```
#include <stdio.h>
#include <string.h>
int main() {
    char s[100];
    int i;
    printf("\nEnter a string : ");
    gets(s);
    for (i = 0; s[i]!='\0'; i++) {
        if(s[i] >= 'a' && s[i] <= 'z') {
            s[i] = s[i] -32;
        }
    }
    printf("\nString in Upper Case = %s", s);
    return 0;
}
```

Anatomy of a Stack Frame

```
; push call arguments, in reverse
push    3
push    2
push    1
call    callee      ; call subroutine 'callee'

callee:
push    ebp        ; save old call frame
mov     ebp, esp   ; initialize new call frame
...do stuff...
mov     esp, ebp
pop    ebp        ; restore old call frame
ret

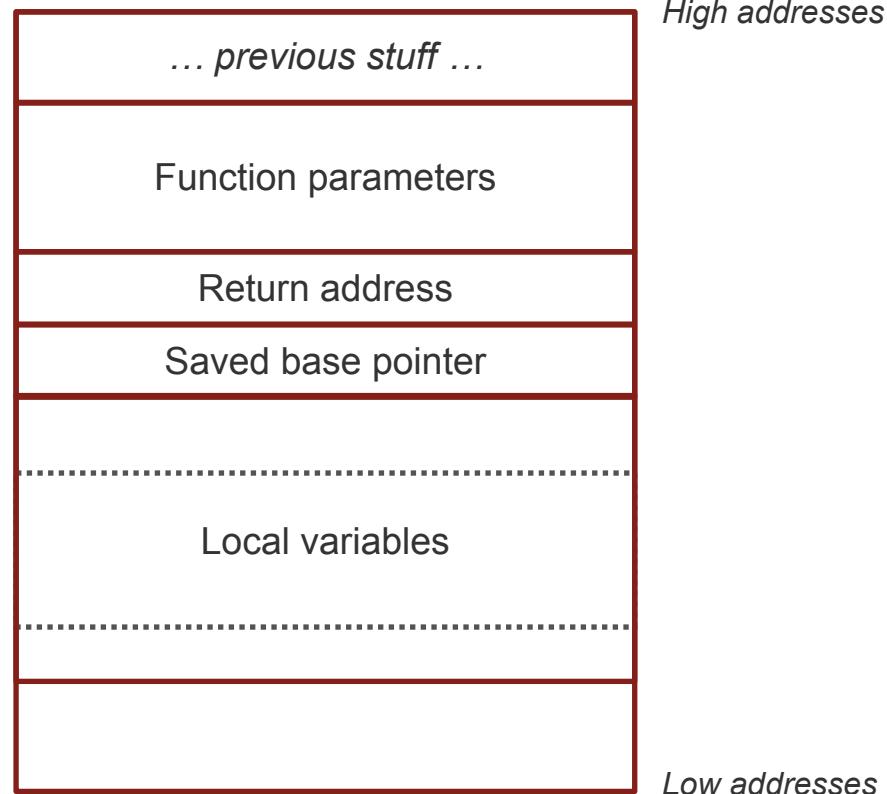
add    esp, 12    ; remove call arguments from frame
```



Anatomy of a Stack Frame

```
; push call arguments, in reverse
push    3
push    2
push    1
call    callee      ; call subroutine 'callee'

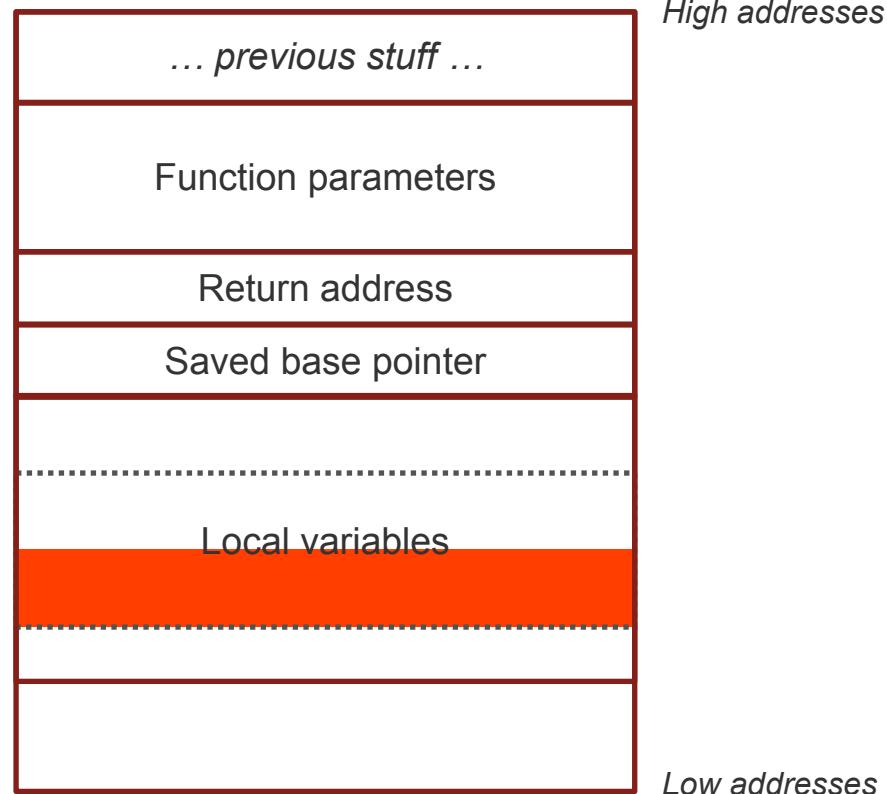
callee:
push    ebp          ; save old call frame
mov     ebp, esp    ; initialize new call frame
...do stuff...
```



Anatomy of a Stack Frame

```
; push call arguments, in reverse
push    3
push    2
push    1
call    callee      ; call subroutine 'callee'

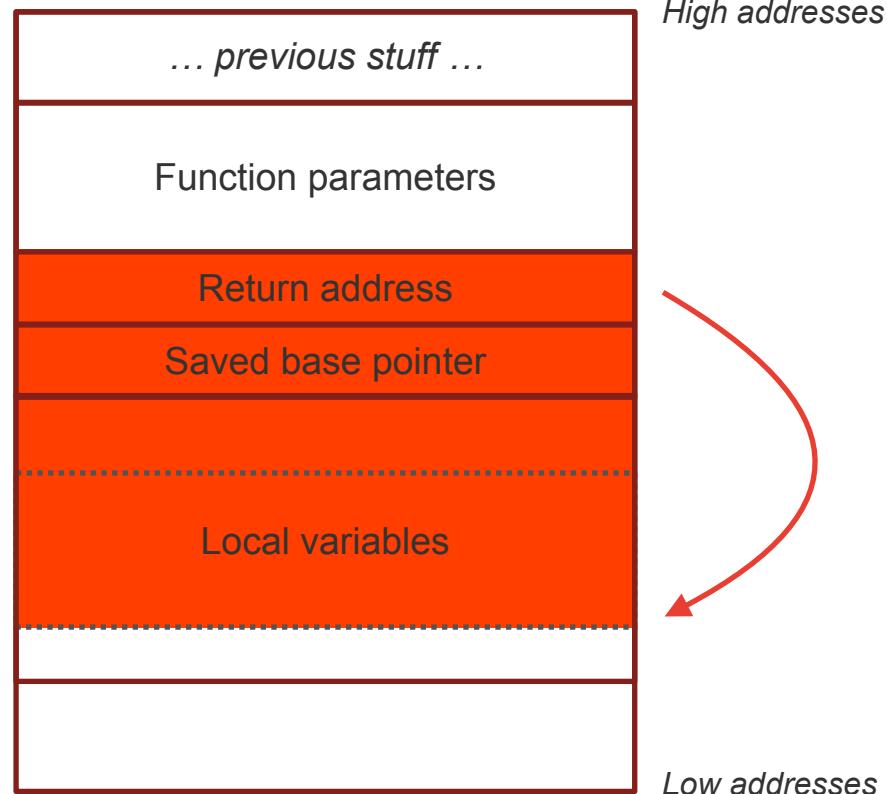
callee:
push    ebp          ; save old call frame
mov     ebp, esp    ; initialize new call frame
...do stuff...
```



Anatomy of a Stack Frame

```
; push call arguments, in reverse
push    3
push    2
push    1
call    callee      ; call subroutine 'callee'

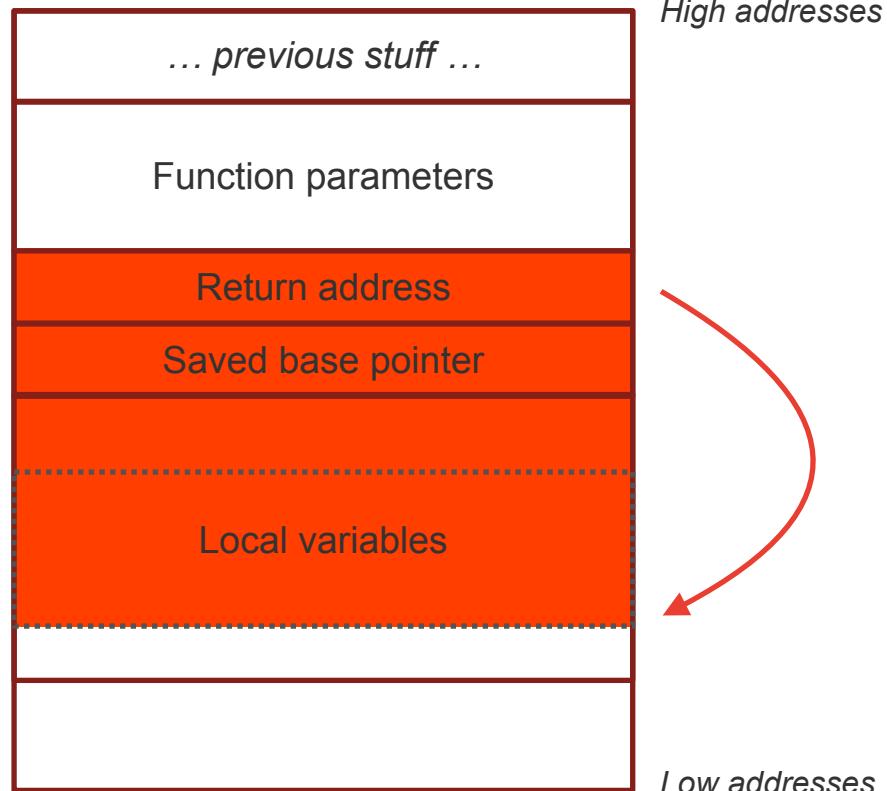
callee:
push    ebp          ; save old call frame
mov     ebp, esp    ; initialize new call frame
...do stuff...
```



Anatomy of a Stack Frame

```
; push call arguments, in reverse
push    3
push    2
push    1
call    callee      ; call subroutine 'callee'

callee:
push    ebp        ; save old call frame
mov     ebp, esp   ; initialize new call frame
...do stuff...
mov     esp, ebp
pop    ebp        ; restore old call frame
ret
```

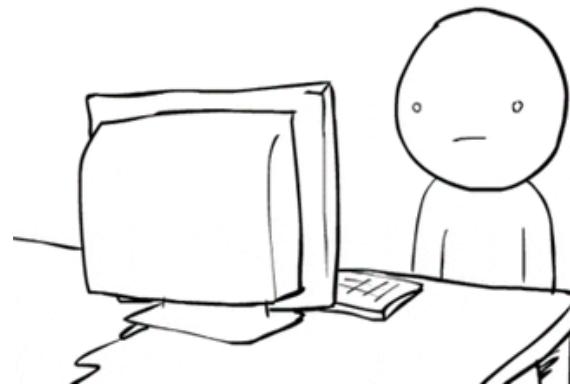


Morris Worm (circa 1988)

```
int main(int argc, char *argv[]) {
    char line[512];
    struct sockaddr_in sin;
    int i, p[2], pid, status;
    i = sizeof (sin);
    if (getpeername(0, &sin, &i) < 0) fatal(argv[0], "getpeername");
    if (gets(line) == NULL) exit(1);
    register char *sp = line;
    ...
    if ((pid = fork()) == 0) {
        close(p[0]);
        if (p[1] != 1) {
            dup2(p[1], 1);
            close(p[1]);
        }
        execv("/usr/ucb/finger", av);
        _exit(1);
    }
    ...
}
```

“Convert a String to Uppercase in C,” circa 2021

```
#include <stdio.h>
#include <string.h>
int main() {
    char s[100];
    int i;
    printf("\nEnter a string : ");
    gets(s);
    for (i = 0; s[i]!='\0'; i++) {
        if(s[i] >= 'a' && s[i] <= 'z') {
            s[i] = s[i] -32;
        }
    }
    printf("\nString in Upper Case = %s", s);
    return 0;
}
```



Okay, well, I'm smarter than that.

Professional engineers don't make such silly mistakes, right?

Comprehensive Experimental Analyses of Automotive Attack Surfaces

Stephen Checkoway, Damon McCoy, Brian Kantor,
Danny Anderson, Hovav Shacham, and Stefan Savage
University of California, San Diego

Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno
University of Washington

Abstract

Modern automobiles are pervasively computerized, and hence potentially vulnerable to attack. However, while previous research has shown that the *internal* networks within some modern cars are insecure, the associated threat model—requiring *prior physical access*—has

This situation suggests a significant gap in knowledge, and one with considerable practical import. To what extent are external attacks possible, to what extent are they practical, and what vectors represent the greatest risks? Is the etiology of such vulnerabilities the same as for desktop software and can we think of defense in the same

“Like many modern cars, our car’s cellular capabilities facilitate a variety of safety and convenience features (e.g. the car can automatically call for help if it detects a crash). However, long-range communication channels also offer an obvious target for potential attackers...”

The car has a 3G modem, but 3G service isn’t available everywhere (this was especially true in 2011, when the paper was written). As such, the car also has an analog audio modem with an associated telephone number! *“To synthesize a digital channel in this environment, the manufacturer uses Airbiquity’s aqLink software modem to convert between analog waveforms and digital bits.”*

“As mentioned earlier, the aqLink code explicitly supports packet sizes up to 1024 bytes. However, the custom code that glues aqLink to the Command program assumes that packets will never exceed 100 bytes or so (presumably since well-formatted command messages are always smaller)”

“We also found that the entire attack can be implemented in a completely blind fashion — without any capacity to listen to the car’s responses. Demonstrating this, we encoded an audio file with the modulated post-authentication exploit payload and loaded that file onto an iPod. By manually dialing our car on an office phone and then playing this “song” into the phone’s microphone, we are able to achieve the same results and compromise the car.”

<http://www.autosec.org/pubs/cars-usenixsec2011.pdf>

Umm... Well I just won't work for a car company?

[Join Extra Crunch](#)[How to
concern
recogn](#)[Login](#)[Is remo
normal](#)[Search](#)[While n
compa
data, o](#)[TC Early Stage
2021](#)[Startups](#)[Videos](#)[Mozilla
COVID-
project](#)[Audio](#)[Newsletters](#)[Extra Crunch](#)[The TC List](#)[Advertise](#)[Events](#)

Apple releases iPhone, iPad and Watch security patches for zero-day bug under active attack

Zack Whittaker @zackwhittaker 1:45 PM PDT • March 27, 2021





CVE List

CNAs

WGs

Board

About

News & Blog

NVD

Go to for:

[CVSS Scores](#)

[CPE Info](#)

[Advanced Search](#)

Search CVE List

Download CVE

Data Feeds

Request CVE IDs

Update a CVE Entry

TOTAL CVE Entries: [133423](#)

HOME > CVE > SEARCH RESULTS

Search Results

There are **10635** CVE entries that match your search.

Name	Description
CVE-2020-9760	An issue was discovered in WeeChat before 2.7.1 (0.3.4 to 2.7 are affected). When a new IRC message 005 is received with longer nick prefixes, a buffer overflow and possibly a crash can happen when a new mode is set for a nick.
CVE-2020-9552	Adobe Bridge versions 10.0 have a heap-based buffer overflow vulnerability. Successful exploitation could lead to arbitrary code execution.
CVE-2020-9535	fmwlanc on D-Link DIR-615Jx10 devices has a stack-based buffer overflow via the formWlanSetup_Wizard webpage parameter when f_radius_ip1 is malformed.
CVE-2020-9534	fmwlanc on D-Link DIR-615Jx10 devices has a stack-based buffer overflow via the formWlanSetup webpage parameter when f_radius_ip1 is malformed.
CVE-2020-9366	A buffer overflow was found in the way GNU Screen before 4.8.0 treated the special escape OSC 49. Specially crafted output, or a special program, could corrupt memory and crash Screen or possibly have unspecified other impact.
CVE-2020-9067	There is a buffer overflow vulnerability in some Huawei products. The vulnerability can be exploited by an attacker to perform remote code execution on the affected products when the affected product functions as an optical line terminal (OLT). Affected product versions include:SmartAX MA5600T versions V800R013C10, V800R015C00, V800R015C10, V800R017C00, V800R017C10, V800R018C00, V800R018C10; SmartAX MA5800 versions V100R017C00, V100R017C10, V100R018C00, V100R018C10, V100R019C10; SmartAX EA5800 versions V100R018C00, V100R018C10, V100R019C10.
CVE-2020-8962	A stack-based buffer overflow was found on the D-Link DIR-842 REVC with firmware v3.13B09 HOTFIX due to the use of strcpy for LOGINPASSWORD when handling a POST request to the /MTFWU endpoint.
CVE-2020-8955	irc_mode_channel_update in plugins/irc/irc-mode.c in WeeChat through 2.7 allows remote attackers to cause a denial of service (buffer overflow and application crash) or possibly have unspecified other impact via a malformed IRC message 324 (channel mode).
CVE-2020-8874	This vulnerability allows local attackers to escalate privileges on affected installations of Parallels Desktop 15.1.2-47123. An attacker must first obtain the ability to execute high-privileged code on the target guest system in order to exploit this vulnerability. The specific flaw exists within the xHCI component. The issue results from the lack of proper validation of user-supplied data, which can result in an integer overflow before allocating a buffer. An attacker can leverage this vulnerability to escalate privileges and execute code in the context of the hypervisor. Was ZDI-CAN-10032.
CVE-2020-8608	In libslirp 4.1.0, as used in QEMU 4.2.0, tcp_subr.c misuses snprintf return values, leading to a buffer overflow in later code.
CVE-2020-8597	eap.c in pppd in ppp 2.4.2 through 2.4.8 has an rhostname buffer overflow in the eap_request and eap_response functions.
CVE-2020-8170	An issue was discovered in Oracle Java 11.0.1. It is possible to overflow the stack when calling the <code>System.out.println</code> function. This could lead to a denial of service or information disclosure.

```

void ares_create_query(const char *name, int dnsclass)
{
    unsigned char *q;
    const char *p;

    /* Compute the length of the encoded name so we can check buflen. */
    int len = 0;
    for (p = name; *p; p++)
    {
        if (*p == '\\\' && *(p + 1) != 0)
            p++;
        len++;
    }
    /* If there are n periods in the name, there are n + 1 labels, and
     * thus n + 1 length fields, unless the name is empty or ends with a
     * period. So add 1 unless name is empty or ends with a period.
     */
    if (*name && *(p - 1) != '.') false if name ends with \.
    len++;

    /* +1 for dnsclass below */
    q = malloc(len + 1);

    while (*name)
    {
        *q++ = /* ... label length, calculation omitted for brevity */
        for (p = name; *p && *p != '.'; p++)
        {
            if (*p == '\\\' && *(p + 1) != 0)
                p++;
            *q++ = *p;
        }

        /* Go to the next label and repeat, unless we hit the end. */
        if (!*p)
            break;
        name = p + 1;
    }

    *q = dnsclass & 0xff; overflows one byte
}

```

One-byte overflow in Chrome OS:

<https://googleprojectzero.blogspot.com/2016/12/chrome-os-exploit-one-byte-overflow-and.html>

Spot the overflow

```
char buffer[128];
int bytesToCopy = packet.length;
if (bytesToCopy < 128) {
    strncpy(buffer, packet.data, bytesToCopy);
}
```

Spot the overflow

```
char buffer[128];
int bytesToCopy = packet.length;
if (bytesToCopy < 128) { ✓ Proper bounds check
    strncpy(buffer, packet.data, bytesToCopy);
}   ✓ Use of strncpy (avoiding unsafe strcpy)
```

Spot the overflow

Signed

```
char buffer[128];
int bytesToCopy = packet.length;
if (bytesToCopy < 128) {
    strncpy(buffer, packet.data, bytesToCopy);
}
```

Cast to size_t (unsigned)

How can we find/prevent these problems?

This is the topic of this whole class :)

Dynamic analysis

- Run the program, watch what it does, and look for problematic behavior
- Can find problems, but only if the program exhibits problematic behavior on the inputs you use to test

“Convert a String to Uppercase in C”

```
#include <stdio.h>
#include <string.h>
int main() {
    char s[100];
    int i;
    printf("\nEnter a string : ");
    gets(s);
    for (i = 0; s[i]!='\0'; i++) {
        if(s[i] >= 'a' && s[i] <= 'z') {
            s[i] = s[i] -32;
        }
    }
    printf("\nString in Upper Case = %s", s);
    return 0;
}
```

Dynamic analysis

- Run the program, watch what it does, and look for problematic behavior
- Can find problems, but only if the program exhibits problematic behavior on the inputs you use to test
- Commonly combined with techniques to run the program with lots of different test inputs (e.g. fuzzing), yet this still can't give us any assurances that code is bug-free

Static analysis

- Read the source code and find problematic parts
- Easy in simple cases (e.g. you can raise an error if anyone ever calls `gets()`)
- Impossible in the general case (Halting Problem)

Static analysis and the Halting Problem

```
int main() {
    // this is not valid syntax but just ignore that
    int a = b = c = d = e = f = rand();
    while (true) {
        a = b * c;
        c = a;
        e = f * 2;
        f = a + b + c;
        d = b / c;

        if (a == 1 && b == 2 && c == 3 && d == 4 && e == 5 && f == 6) {
            // exit
            return 0;
        }
    }
}
```

Static analysis and the Halting Problem

```
int main() {
    // this is not valid syntax but just ignore that
    int a = b = c = d = e = f = rand();
    while (true) {
        a = b * c;
        c = a;
        e = f * 2;
        f = a + b + c;
        d = b / c;

        if (a == 1 && b == 2 && c == 3 && d == 4 && e == 5 && f == 6) {
            // segfault
            *(int*)(NULL) = 0;
        }
    }
}
```

Static analysis

- Read the source code and find problematic parts
- Easy in simple cases (e.g. you can raise an error if anyone ever calls `gets()`)
- Impossible in the general case (Halting Problem)
- Are there ways that we can make static analysis more tractable/helpful?
(Topic of Thursday's lecture and next week)

About CS 110L 🙌

Course outline

- Key question: How can we prevent common mistakes in systems programming?
 - This is not a Rust class, although almost all of our programming will be done in Rust
 - How do we find and prevent common mistakes in C/C++?
 - How does Rust's type system prevent common memory safety errors?
 - How do you architect good code?
 - Avoiding multiprocessing pitfalls
 - Avoiding multithreading pitfalls
 - Putting all of this into practice: Networked systems

Course outline

- Corequisite: CS 110
- Pass/fail
 - You will get out what you put in
- Components:
 - Lecture
 - Weekly exercises (40%)
 - Two projects (40%)
 - Participation (20%)

Projects

- Project 1: Mini GDB
- Project 2: High-performance web server
- Functionality grading only
 - The Rust compiler will be your interactive style grader!
- These projects are intended to give you additional experience in building real systems, while having to think about some of the safety issues we're discussing
- Have a different idea? Let us know!

Exercises

- Each week, we'll give you some small programming problems to reinforce the week's lecture material
- Alternatively: Write a blog post about something you're learning in the class
- Expected time: 1-3 hours
- In addition, you'll be asked to complete an anonymous survey about how the class is going and how we can improve

Work for Thursday

Before class, spend 10 minutes trying to spot as many bugs as you can find in this code snippet:

<https://web.stanford.edu/class/cs110l/lecture-notes/lecture-02/>

(From the course website, click “Lecture notes” under Lecture 2)