Welcome to CS 110L 👋

Ryan Eberhardt and Armin Namavari
April 7, 2020

Who are we?

Armin Namavari

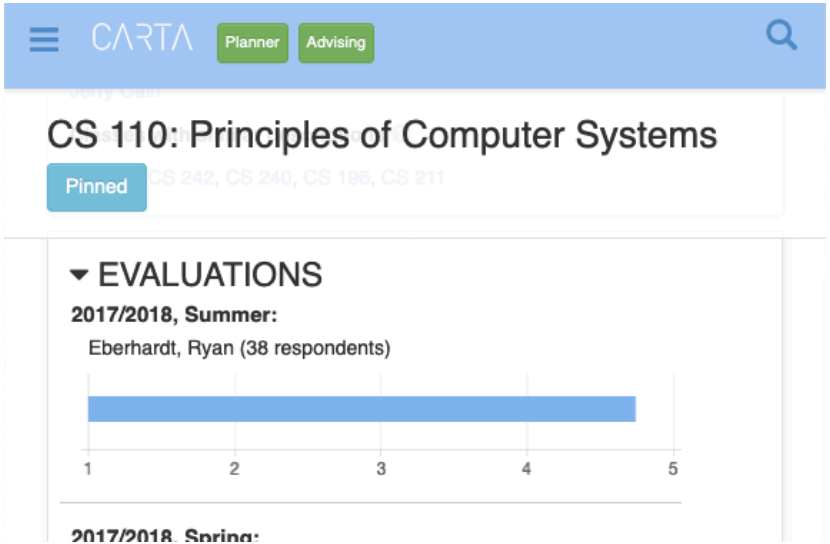
- Coterm (class of '19 undergrad)
- Interested in security/building secure systems, applied cryptography, theory
- I've used Rust in the context of research on Tock, an embedded OS
- During shelter-in-place I've been...
  - been learning how to longboard
  - finger-knitting a blanket
  - trying to cook new things
- I like climbing and playing ultimate!

Ryan Eberhardt

- Coterm focused on systems and security
- I like growing things

Ryan Eberhardt

- Coterm focused on systems and security
- I like growing things
- Pretty into music, especially funk, jazz, and fusion
- I love doing pottery
- Complete Rust impostor
- But I do know CS 110 pretty well



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

Who are you?

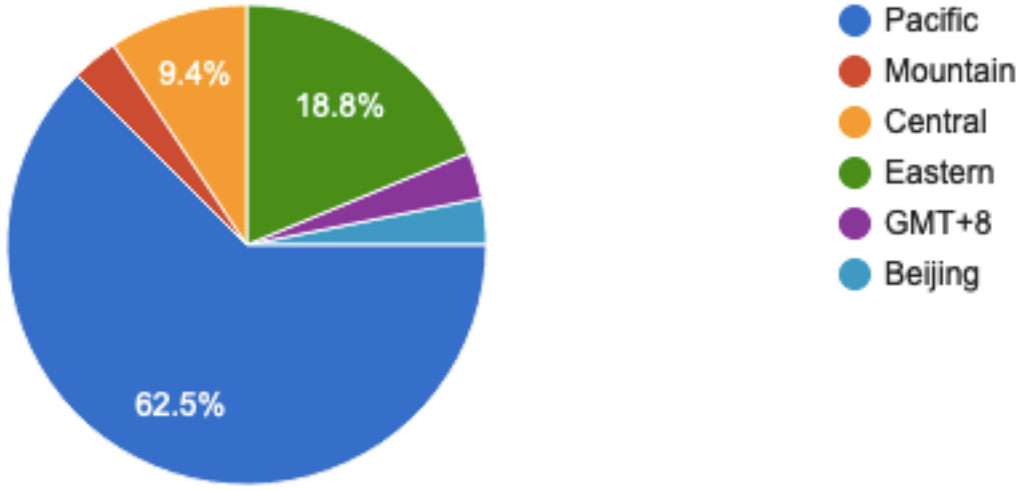
Who are you?

Fun and quirky community of 33 registered (as of Monday) + a few auditors!

Who are you?

What time zone are you in?

32 responses



Who are you?

Why are you taking this class?

- *I want to learn Rust!*
- *Enhance what I will learn in CS110*
- *I'm growing to love systems, and I hate errors. CS 110L says it'll help me with this*
- *I am developing more interest in maintaining secure code, particularly in low-level systems, so this course seems like it'd be great for me.*
- *The projects look super cool! Also, in general, I think systems is really difficult for me, but despite this, I genuinely thought the content of 107 was really interesting and thus I think it'd be great for me to be able to explore these topics more deeply.*

Who are you?

Have you heard anything about Rust before?

- Most people: *“Nope.”*
- Note: If you have taken CS 242 (two 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.

32 responses

I made fried chicken sandwiches for the first time yesterday and they were delicious.

I may or may not have mapped parts of the steam tunnels

I'm on Stanford Jump Rope! :D

I can lick my nose

I love motorycles

I'm an archer!

I enjoy climbing redwood trees

I play squash (which is a sport as well as a vegetable)

Obsessed with indie music! I want to be a musician some day.

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

Why Rust?

Why Rust?

Why not C/C++?

Why not GC’ed languages (Java, Python, Go, etc.)

Why not C/C++?

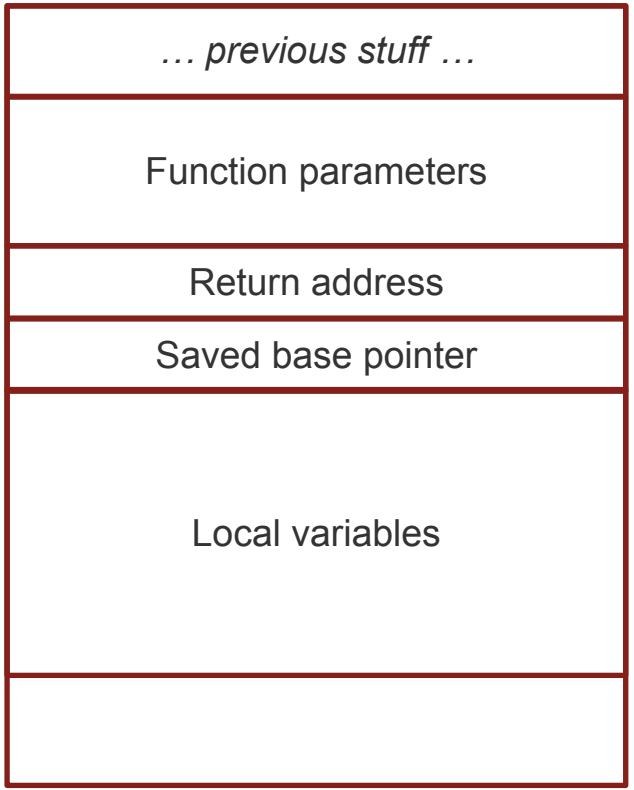
(topic of Thursday's lecture)

“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

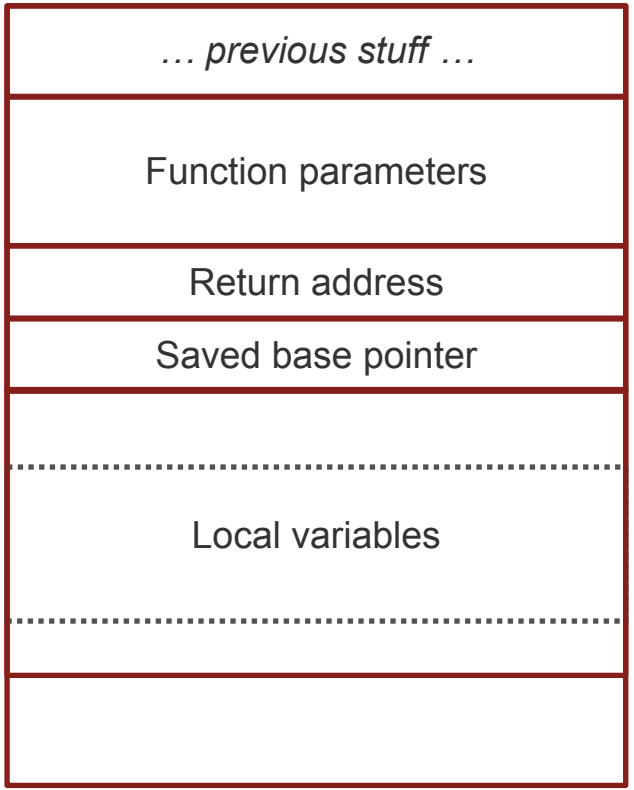
```
; push call arguments, in reverse
push    3
push    2
push    1
call    callee    ; call subroutine 'callee'

    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               ; return


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



From https://en.wikipedia.org/wiki/X86_calling_conventions#cdecl

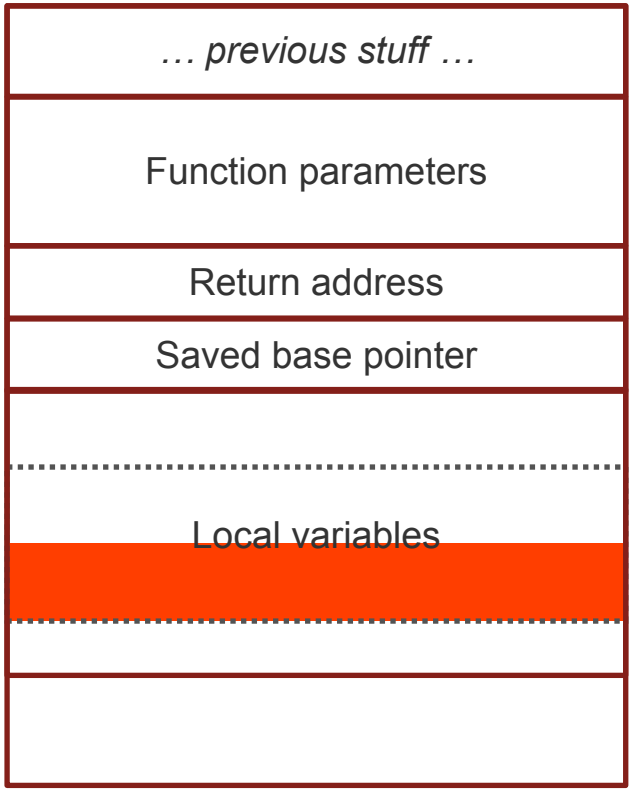
Anatomy of a Stack Frame

```
; 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...
```



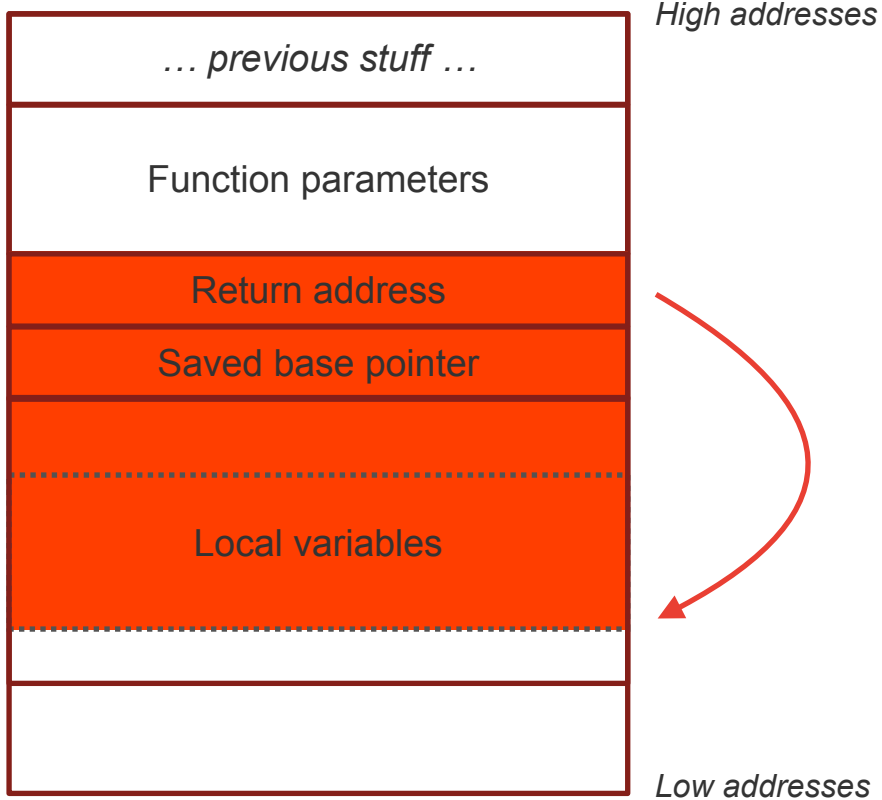
Anatomy of a Stack Frame

```
; 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...
```



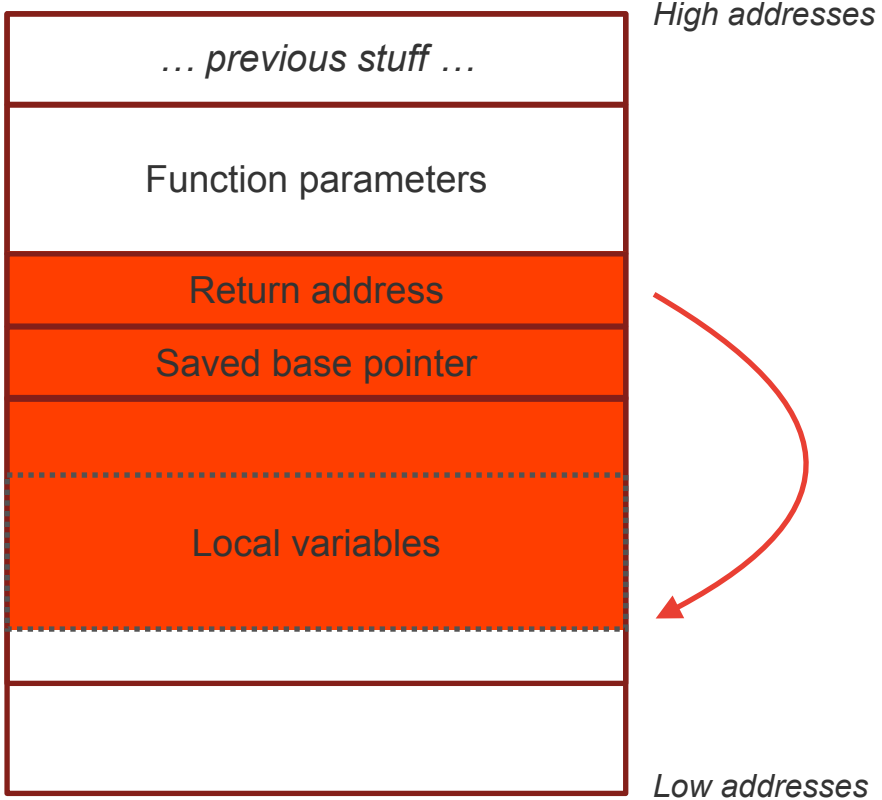
Anatomy of a Stack Frame

```
; 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               ; return
```

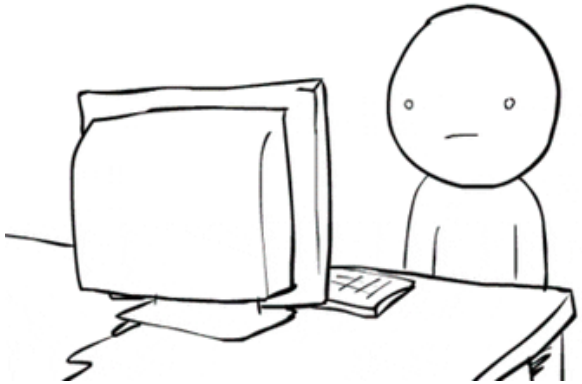


Morris Worm (circa 1988)

```
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 2020

```
#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 covert 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?

Google Squashes High-Severity Flaws in Chrome Browser

Author:
Lindsey O'Donnell

April 2, 2020 / 5:19 pm

minute read

Write a comment

INFOSEC INSIDER

A Practical Guide to Zero-
Security
January 15, 2020

7 Tips for Maximizing You
December 31, 2019

Mean Time to Hardening:
Gen Security Metric
December 30, 2019

Combining AI and Playbo
Predict Cyberattacks
December 26, 2019

The Case for Cyber-Risk

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	fmwlan.c 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	fmwlan.c 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.

```c
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) != '.')
    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;
}
```

false if name ends with \.

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) {
    strncpy(buffer, packet.data, bytesToCopy);
}
```

✓ Proper bounds check

✓ 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)

More reasons to come on Thursday

Aside: doesn’t Valgrind tell you about these things?

```
==1234== Memcheck, a memory error detector
==1234== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==1234== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==1234== Command: ./poop
==1234==
==1234== Invalid write of size 8
==1234==    at 0x400BCF: poop (main.c:24)
==1234==    by 0x400CCC: plop (main.c:100)
==1234==    by 0x400DFF: main (main.c:200)
==1234==  Address 0x51f25c0 is 16 bytes inside a block of size 20 alloc'd
==1234==    at 0x4C2B6CD: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==1234==    by 0x400BBB: poop (main.c:20)
==1234==    by 0x400CCC: plop (main.c:100)
==1234==    by 0x400DFF: main (main.c:200)
```

Why not use GC’ed languages?

Dear X,

I am looking forward to meeting you, and to a great year in Kimball!

Please consider an idea that I think will make life just a tiny bit better for everyone in the dorm this year.

Last year, a number of us noticed that some people in the dorm were pretty messy, their rooms were a mess, and trash piled up.

It turns out that not only are these trash piles unpleasant, but they can be a hazard, potentially even to others.

According to *The Cardinal Safety Letter* (2012) :

“We have seen some fairly impressive mountains of trash overflowing the little dorm room trash cans. This is not sanitary in the least!”

We also know that exhortations to clean up too often fall on deaf ears.

… more pleas follow …

The good news is that we have a completely painless solution that will be totally inclusive, promote a clean dorm, reduce stress, and engages with Stanford's goal of sustainability.

It's all set to go, just pending your go-ahead.

I will collect the trash from each room in Kimball every week* to help everyone maintain a clean living environment. For less than $.50 per student per weekday, we'll take out everyone's trash for all 10 weeks of the quarter.* By using dorm funds, it doesn't really cost anyone anything, yet we all benefit.

It's a great use of dorm funds, because it'll benefit every member of the dorm equally, which is exactly what dorm funds are for.

It's free for the residents: all we have to do is tie our bags of trash, place them outside our doors by midnight on Sunday, and I'll pick them up on Monday – providing a clean start to the week. Students will be saved the hassle and unpleasantness of completing this tiresome chore, and none of us will have to put up with the messy consequences of piles of trash in dorm rooms.

For just $25 per student, the entire dorm's trash is taken care of for the entire quarter.

A twist

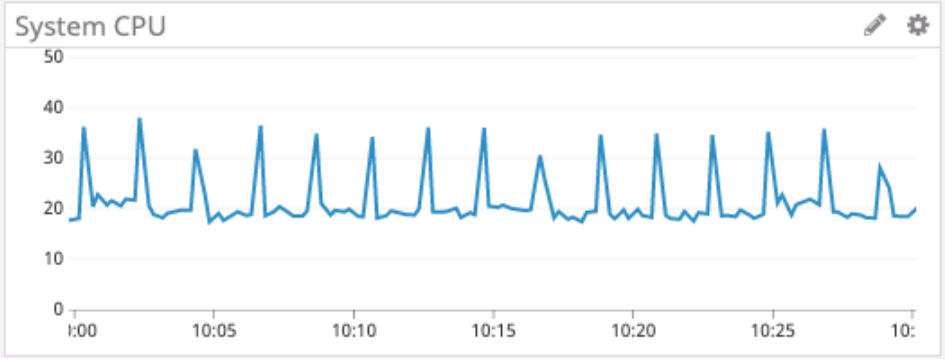
- Instead of putting your trash outside, leave it inside your room
- The GC will come knocking when it’s time to clean up

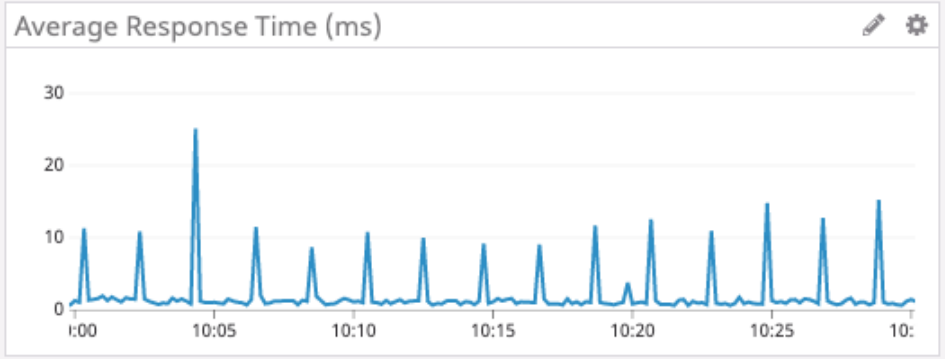
Downsides of garbage collection

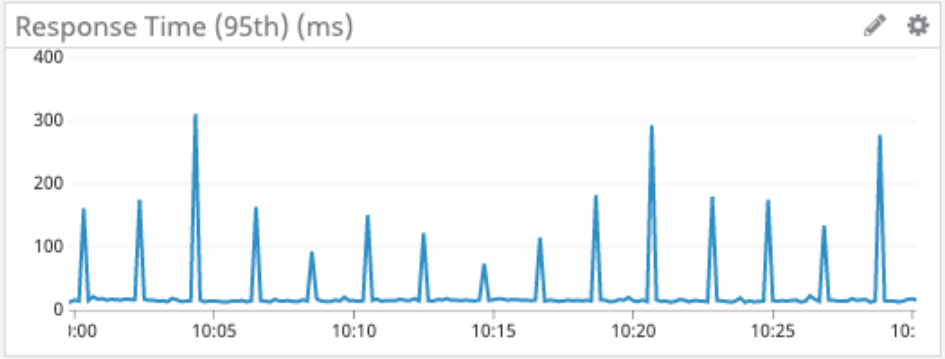
- Expensive
  - No matter what type of garbage collection is used, there will always be nontrivial memory overhead
- Disruptive
  - Drop what you're doing — it's time for GC!
- Non-deterministic
  - When will the next GC pause be? Who knows! Depends on how much memory is being used
- Precludes manual optimization
  - In some situations, you may want to structure your data in memory in a specific way in order to achieve high cache performance
  - GC can't know how you will use memory, so it optimizes for the average use case

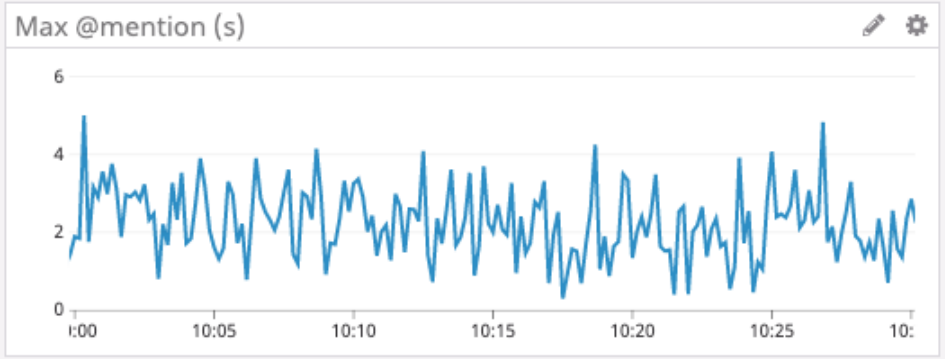
Why Discord is switching from Go to Rust

Jesse Howarth Follow
Feb 4 · 10 min read









Note latency spikes every 2 minutes

LinkedIn Engineering:

“In our production environments, we have seen unexplainable large STW pauses (> 5 seconds) in our mission-critical Java applications.”

https://engineering.linkedin.com/blog/2016/02/eliminating-large-jvm-gc-pauses-caused-by-background-io-traffic

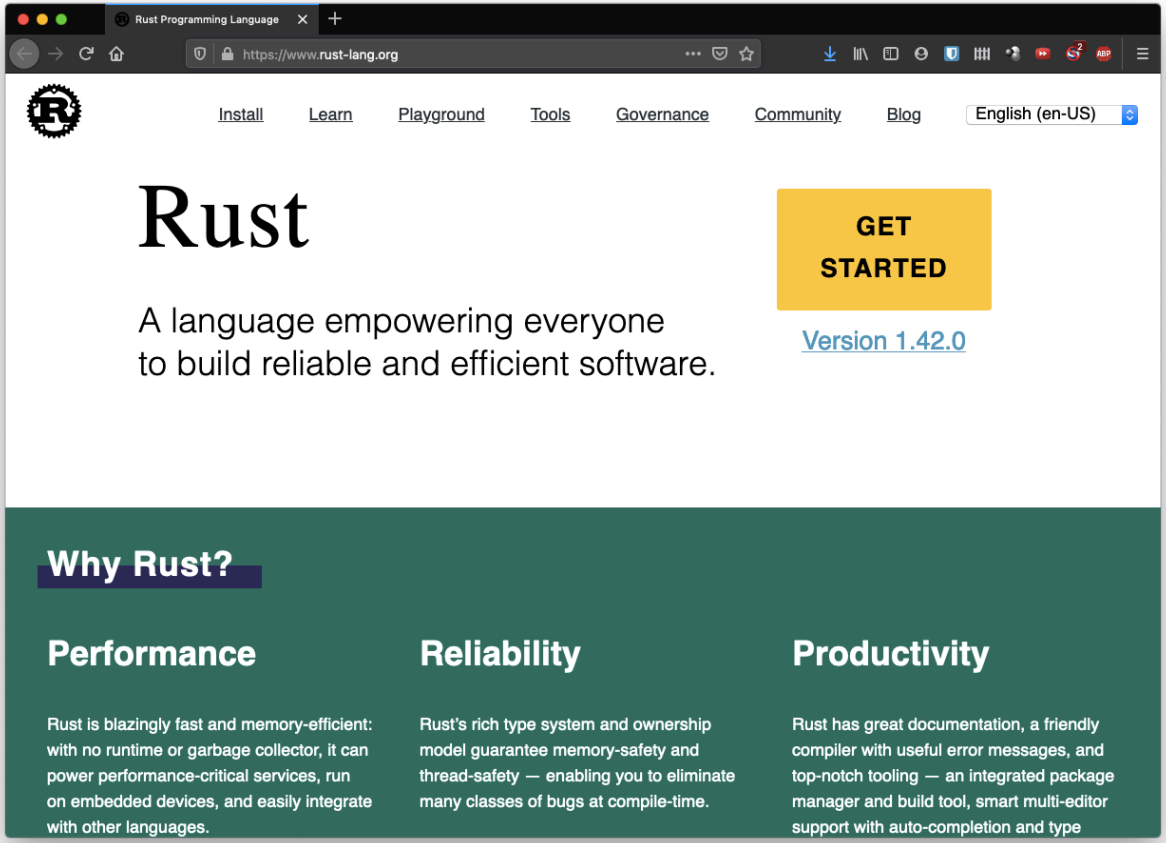
Latency matters

- User interfaces
- Games
- Self-driving cars
- Payment processing
- High frequency trading

Garbage collectors aren’t all about safety

- Later in the quarter, we’ll learn about race conditions
- Garbage collection does not preclude race conditions! Memory safety issues persist even in garbage-collected environments

Design goals of Rust



About CS 110L 👋

Course outline

- Corequisite: CS 110
- Pass/fail
  - You will get out what you put in
- First 3-4 weeks: Safety in a CS 107 context
- Rest of course: Safety in a CS 110 context
- Components:
  - Lecture
  - Weekly exercises (20%)
  - Two projects (60%)
  - 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
- 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

Week 1 Exercise

- The first week, we'll be mainly covering conceptual material about Rust in lecture
- But that's no excuse for you to not start playing around with the language and getting used to its syntax!
- The first exercise will be to implement a simple hangman command line game.
- Our goal is to expose you to some of Rust's syntax without you having to deal with some of its quirks (which we'll discuss in more detail next week).
- You'll probably have to do some of your own searching through docs/stack overflow/etc, but we're available on Slack to support you! (as are your fellow classmates)

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)