
title:

- The Rust Programming Language

author:

- Ben Goldberg

What is This Talk

- ▶ What is Rust?
- ▶ Why would you use Rust?
- ▶ Why wouldn't you use Rust?

What This Talk Isn't

- ▶ Teaching you Rust
 - ▶ In 90 minutes?!?

Learn Rust

- ▶ rust-lang.org
- ▶ CSCI-541/641 Programming Skills in Rust by Prof. Fluet

A Brief History of Programming Languages

In the beginning there was C...

C

Pros:

- ▶ Low level control of memory
 - ▶ Needed for systems programming
- ▶ Simple
- ▶ Stable

Cons:

- ▶ Hard to write complex systems
- ▶ Too easy to mess up memory
- ▶ Parallel computing is **hard**

C++

Pros:

- ▶ Easier to write complex software
 - ▶ We OO now

Cons:

- ▶ Very complex
- ▶ When things go *wrong*, they go **wrong**

Note

- ▶ Modern C++ is better
 - ▶ Because it's doing some of what Rust does

Memory Safety

- ▶ Managing memory is hard
- ▶ Especially is large, complex programs
- ▶ Especially especially when multi-threaded

```
char *str = malloc(10);  
free(str);  
// Bad things happend  
printf("%s", str);
```


Manual Memory Management Isn't Good Enough

- ▶ Memory Leaks
- ▶ Buffer overflow
- ▶ Use after free
- ▶ Double free
- ▶ Null pointer dereference
- ▶ Read uninitialized memory
- ▶ Race conditions

Manual Memory Management Isn't Good Enough

[Home](#) > [Hacking](#) > [Vulnerabilities](#)

BACK TO BASICS

What is the Heartbleed bug, how does it work and how was it fixed?

The mistake that caused the Heartbleed vulnerability can be traced to a single line of code in OpenSSL, an open source code library. Here's what you need to know now.



By **Josh Fruhlinger**

CSO | SEP 13, 2017 2:53 AM PT

Heartbleed is a vulnerability that came to light in April of 2014; it allowed attackers unprecedented access to sensitive information, and it was present on thousands of web servers, including those running major sites like Yahoo.

Heartbleed was caused by a flaw in OpenSSL, an open source code library that implemented the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. In short, a malicious user could easily trick a vulnerable web server into sending sensitive information, including usernames and passwords.

CURRENT JOB LISTINGS

Learn more about SSL/TLS attacks on the internet and how to protect SSL/TLS data

Manual Memory Management Isn't Good Enough

This simple link instantly crashes Google Chrome



By JAMES TEMPERTON

Monday 21 September 2015


**http://a/
%%30%30**

Manual Memory Management Isn't Good Enough

May 29, 2015, 04:18am

Apple Acknowledges Disastrous iPhone Messages Bug, Suggests This Temporary Fix



Amit Chowdhry Contributor 

Tech enthusiast, born in Ann Arbor and educated at Michigan State

Earlier this week, I [wrote about how a new iOS bug emerged](#) that enabled iPhone users to crash another person's iPhone by simply sending a text message. The text message -- which simply says: effective. Power  -- causes the iPhone of the recipient to crash continuously if the text is received while in lock screen mode. The “Effective Power” bug (also known as Unicode of Death) only causes issues between iPhone-to-iPhone communication.

Manual Memory Management Isn't Good Enough

Search Results

There are **9731** CVE entries that match your search.

Name	Description
CVE-2019-7154	The main function in tools/wasm2js.cpp in Binaryen 1.38.22 has a heap-based buffer overflow because Emscripten is misused, triggering an error in cashew::JSPrinter::printAst() in emscripten-optimizer/simple_ast.h. A crafted input can cause segmentation faults, leading to denial-of-service, as demonstrated by wasm2js.
CVE-2019-6991	A classic Stack-based buffer overflow exists in zm_LoadUser() function in zm_user.cpp of the zmu binary in ZoneMinder through 1.32.3, allowing an unauthenticated attacker to execute code via a long username.
CVE-2019-6977	gdImageColorMatch in gd_color_match.c in the GD Graphics Library (aka LibGD) 2.2.5, as used in the Imagecolormatch function in PHP before 5.6.40, 7.x before 7.1.26, 7.2.x before 7.2.14, and 7.3.x before 7.3.1, has a heap-based buffer overflow. This can be exploited by an attacker who is able to trigger imagecolormatch calls with crafted image data.
CVE-2019-6439	examples/benchmark/tls_bench.c in a benchmark tool in wolfSSL through 3.15.7 has a heap-based buffer overflow.
CVE-2019-6250	A pointer overflow, with code execution, was discovered in ZeroMQ libzmq (aka 0MQ) 4.2.x and 4.3.x before 4.3.1. A v2_decoder.cpp zmq::v2_decoder::t::size_ready integer overflow allows an authenticated attacker to overwrite an arbitrary amount of bytes beyond the bounds of a buffer, which can be leveraged to run arbitrary code on the target system. The memory layout allows the attacker to inject OS commands into a data structure located immediately after the problematic buffer (i.e., it is not necessary to use a typical buffer-overflow exploitation technique that changes the flow of control).
CVE-2019-6247	An issue was discovered in Anti-Grain Geometry (AGG) 2.4 as used in SVG++ (aka svggpp) 1.2.3. A heap-based buffer overflow bug in svggpp_agg_render may lead to code execution. In the render_scanlines_aa_solid function, the blend_hline function is called repeatedly multiple times. blend_hline is equivalent to a loop containing write operations. Each call writes a piece of heap data, and multiple calls overwrite the data in the heap.
CVE-2019-1651	A vulnerability in the vContainer of the Cisco SD-WAN Solution could allow an authenticated, remote attacker to cause a denial of service (DoS) condition and execute arbitrary code as the root user. The vulnerability is due to improper bounds checking by the vContainer. An attacker could exploit this vulnerability by sending a malicious file to an affected vContainer instance. A successful exploit could allow the attacker to cause a buffer overflow condition on the affected vContainer, which could result in a DoS condition that the attacker could use to execute arbitrary code as the root user.
CVE-2019-1000006	RIOT RIOT-OS version after commit 7af03ab624db0412c727eed9ab7630a5282e2fd3 contains a Buffer Overflow vulnerability in sock_dns, an implementation of the DNS protocol utilizing the RIOT sock API that can result in Remote code executing. This attack appears to be exploitable via network connectivity.
CVE-2019-0761	In libjpeg-turbo 1.4.0 through 1.5 and 7.0 through 7.1.5, the APJ decompress could result in a heap-based buffer overflow. This issue addressed in jpeg-turbo's socket library by checking for 3

The Solution Up Until Now

Garbage Collectors!

- ▶ Guarantee *some* memory safety by taking away control
- ▶ Reduce performance
- ▶ Multi-threading is still hard
- ▶ Most modern langs use a GC (Java, JS, Python, Go, etc.)

How they work (oversimplified)

- ▶ Every object has a counter that tracks how many references to it there are
- ▶ When the count hits zero, the memory is freed.

GCs Are Good Enough Most of the Time, but Not Always

- ▶ Discord switch from Go to Rust due to inconstant performance from GC pauses
- ▶ Can't write an OS or kernel code*
 - ▶ No Python in Linux
- ▶ Can't target embedded/real-time devices (ex. Arduino)

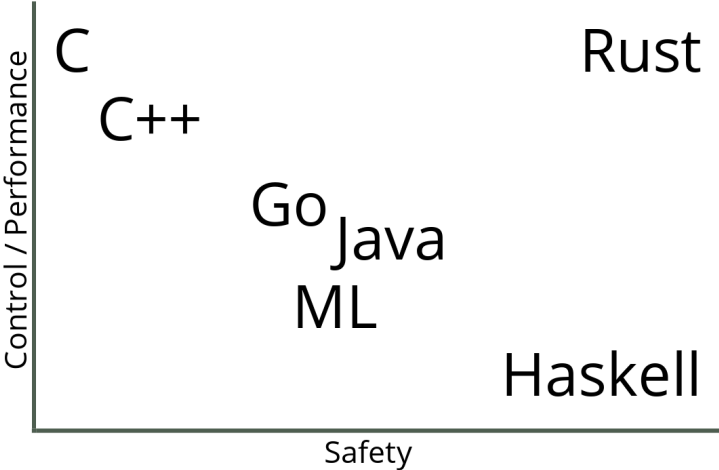
The Trade Off

Safety or Performance

What do we want?

We want to write performant and reliable programs easily and productively

Comparison



What is Rust?

A statically & strongly typed, multi-paradigm language that compiles to machine code. It has uses a novel ownership & borrowing system to manage memory safely and automatically with a garbage collector. Rust has a strong focus on correctness and performance.

What does that mean?

Ahead of Time Compiled

Rust is compiled to machine code ahead of running like C or C++

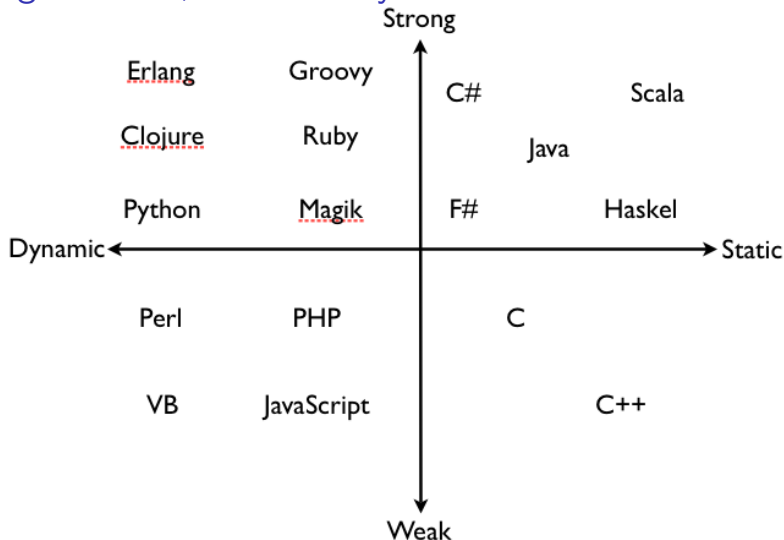
Statically Typed

Every variable has an unchanging type known at compile time

Strongly Typed

The compiler enforces the type system (ex. You can't just cast a number to a string)

Strong vs Weak, Static vs Dynamic



Source: www.josephspurrier.com/strong-weak-dynamic-and-static-typed-programming-languages

Ownership & Borrowing

The main *magic* of Rust

- ▶ Every object is owned by some context
 - ▶ Only the owner can access the object
 - ▶ Ownership can be transferred
- ▶ A context can borrow an object
 - ▶ While borrowed, the owner can't access it
 - ▶ The borrow is scoped to some area of code
 - ▶ Many immutable borrows or one mutable borrow
 - ▶ immutable = read-only
 - ▶ mutable = read/write

Read/Write safety

You can have many readers **or** one writer, safely

What Does This Look Like?

Let's learn *a little* Rust

If you want to play with Rust in your browsers go to play.rust-lang.org

Hello World

```
fn main() {  
    println!("Hello, world!");  
}
```

Immutable by default

All variables are immutable by default

Doesn't work

```
let x = 5;
```

```
x = 3;
```

Works

```
let mut x = 5;
```

```
x = 3;
```

Static Typing

All variables must have one type

```
let x: i32 = 77;
```

But with type inference

```
let x = 77;
```

Rust's Core Principle

Aliasing XOR Mutation

Ownership

Ownership rules

- ▶ Each value in Rust has a variable that's called its owner
- ▶ There can only be one owner at a time
- ▶ When the owner goes out of scope, the value will be dropped

```
fn main() {  
    let x = 1;  
    {  
        let y = 5;  
        println!("x:{}", y:{}", x, y);  
    }  
    // Doesn't compile!!!!  
    println!("x:{}", y:{}", x, y);  
}
```

Another Example

```
fn hello(name: String) {  
    println!("Hello {}!", name);  
    // name is destroyed here  
}  
  
fn main() {  
    let name = String::from("RIT LUG");  
    hello(name);  
    // Doesn't compile because name has been freed  
    println!("Goodbye {}", name);  
}
```


References and Borrowing

We can lend out ownership of a value with a reference

```
fn hello(name: &str) {  
    println!("Hello {}!", name);  
}
```

```
fn main() {  
    let name = "RIT LUG";  
    hello(&name);  
    println!("Goodbye {}", name);  
}
```

Immutable vs Mutable References

Immutable reference

```
// Doesn't compile  
fn inc(x: &i32) {  
    x += 1;  
}
```

Mutable reference

```
fn inc(x: &mut i32) {  
    x += 1;  
}  
  
fn main() {  
    let mut x = 1;  
    inc(&mut x);  
}
```

Immutable vs Mutable References cont.

Aliasing or Mutability

```
let mut v1 = 3;  
let r1 = &v1;  
let r2 = &v1;
```

```
// Doesn't compile  
let mut v2 = 4;  
let r1 = &mut v2;  
let r2 = &mut v2;
```

Actually this does compile because Rust is smart, but it wouldn't if you tried to actually use both mutable references

Rust Prevents Many Classes of Memory Errors

- ▶ Use after free
- ▶ Null pointer dereference
- ▶ Using uninitialized memory
- ▶ Double free
- ▶ Buffer overflow
- ▶ Data race
- ▶ Many concurrency/multi-threading bugs
 - ▶ Rust also enforces memory safety across threads, which most GCs don't
- ▶ Type errors

If it compiles it probably works™

Rust is *Fast*

- ▶ Rust's speed is on par with C and C++

Cargo

Rust has a built tool/package manager called cargo that's very good

- ▶ Makes it very easy to install libraries
- ▶ There's a single blessed PM that everyone uses unlike Python
- ▶ Run `cargo build` and it just works™

Use Cases

- ▶ Command line tools
- ▶ Operating systems
- ▶ Network services
- ▶ Web Apps
- ▶ Webassembly
- ▶ Embedded

Companies Using Rust in Production

- ▶ Mozilla
 - ▶ Parts of Firefox
- ▶ Facebook
- ▶ Dropbox
 - ▶ Storage backend
- ▶ Cloudflare
- ▶ Discord
 - ▶ Parts of backend
- ▶ NPM
- ▶ Yelp
- ▶ Tilde

The Down Sides

Rust sounds great. . . but what's the catch?

The Down Sides

- ▶ The compiler is *slow*
 - ▶ Medium sized Rust programs can take several minutes to compile
- ▶ Many Rust programmes have a lot of dependencies
 - ▶ Making install and publishing libraries has downsides
 - ▶ The NPM problem
- ▶ Hard to learn
 - ▶ Rust is pretty different to most modern langs
- ▶ Not a lot of jobs, especially entry level
- ▶ Many features still in development (async/await)
 - ▶ Leads to a lot of ecosystem churn
- ▶ No LTS compiler releases
 - ▶ Makes packaging Rust programs for LTS Linux distros harder
- ▶ Only one real compiler implementation
- ▶ Doesn't support as many CPU architectures and OSes as C

The Down Sides

All of the problems are being worked on but for now, this is how it is

The End

Questions?

Other Stuff

- ▶ These slides are licensed under the CC-BY-SA license
- ▶ Source code available at git.sr.ht/~zethra/rust_presentation