The Rust Programming Language

Ben Goldberg

Rust

A Programming Language for the Future

Thank You

Thank you to rust-lang.org for providing content and examples!

Overview

- The sales pitch
- Use cases
- The Rust language

Other Languages

What's wrong with other languages?

Other Languages: Python

What's wrong with Python?

Other Languages: Python

- Slow
- Heavy
- Doesn’t catch mistakes

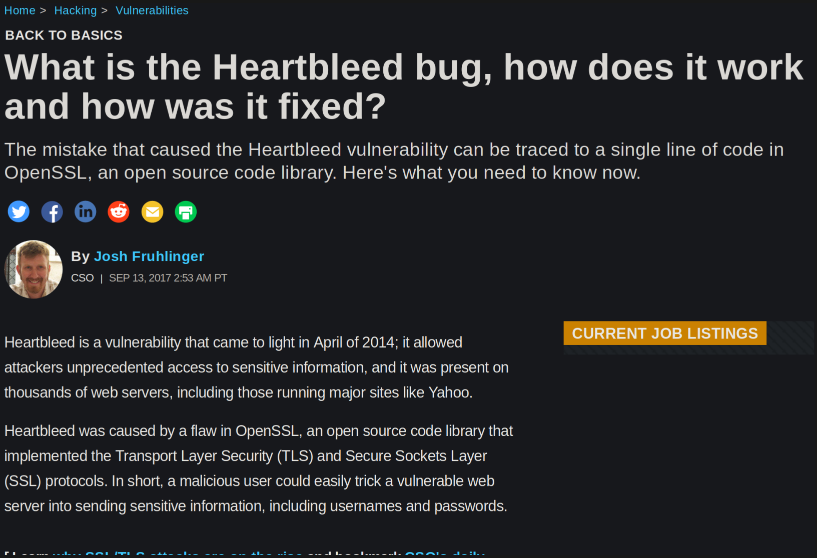
Other Languages: C/C++

What's wrong with C/C++?

Other Languages: C/C++

- Memory Leaks
- Buffer overflow
- Use after free
- Double free
- Null pointer dereference
- Read uninitialized memory
- Race conditions
- No good build tools for large projects

Other Languages: C/C++



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

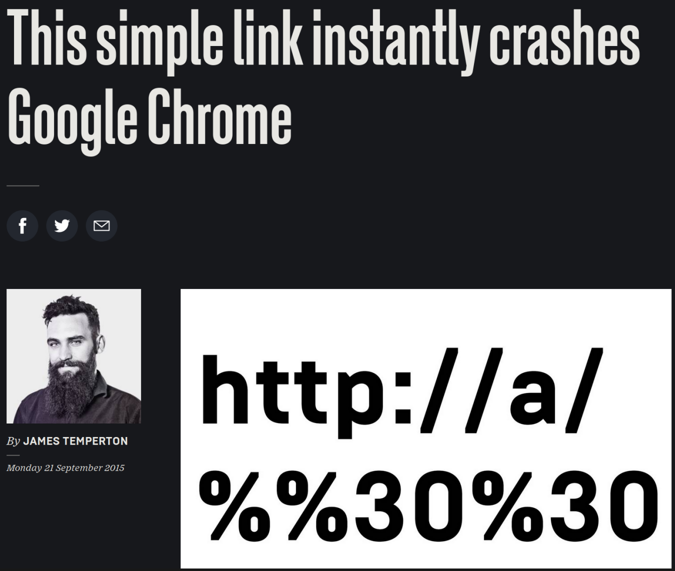
Heartbleed is a vulnerability that came to light in April of 2014; it allowed attackers unprecedented access to sensitive information, and it was present on thousands of web servers, including those running major sites like Yahoo.

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.

[Learn why SSL/TLS attacks are on the rise and bookmark CSO's daily

Figure 1: Heartbleed

Other Languages: C/C++



This simple link instantly crashes Google Chrome

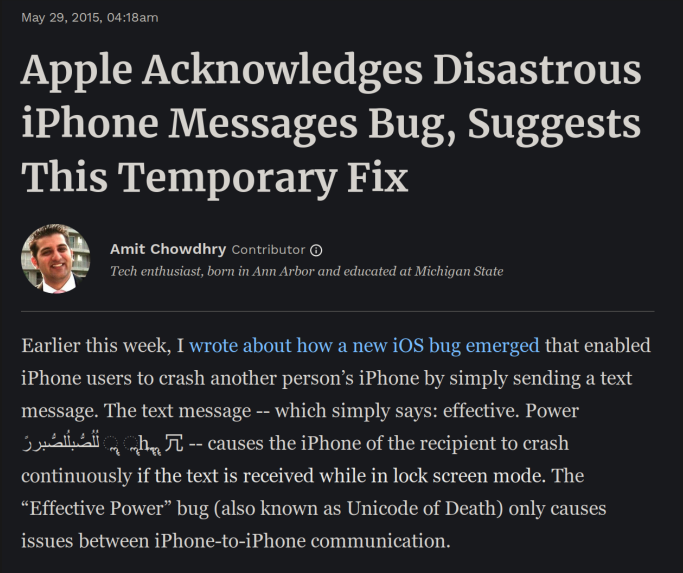
By **JAMES TEMPERTON**

Monday 21 September 2015

http://a/%%30%30

Figure 2: Chrome URL

Other Languages: C/C++



May 29, 2015, 04:18am

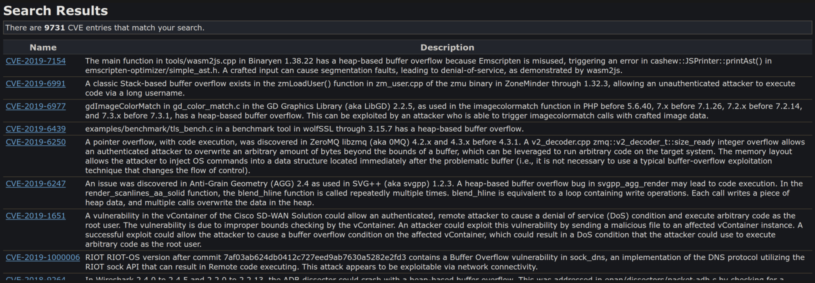
Apple Acknowledges Disastrous iPhone Messages Bug, Suggests This Temporary Fix

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 لُلُصّبُلُلصّبُررً ॣ ॣh ॣ ॣ 冗 -- 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.

Figure 3: Effective Power

Other Languages: C/C++



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 the zmLoadUser() 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 svgpp) 1.2.3. A heap-based buffer overflow bug in svgpp_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-2018-9264	In Wireshark 2.4.0 to 2.4.5 and 2.2.0 to 2.2.13, the ADB dissector could crash with a heap-based buffer overflow. This was addressed in epan/dissectors/packet-adb.c by checking for a

Figure 4: Buffer Overflow

What do we want?

We want to write performant and reliable programs easily and productively

Comparison

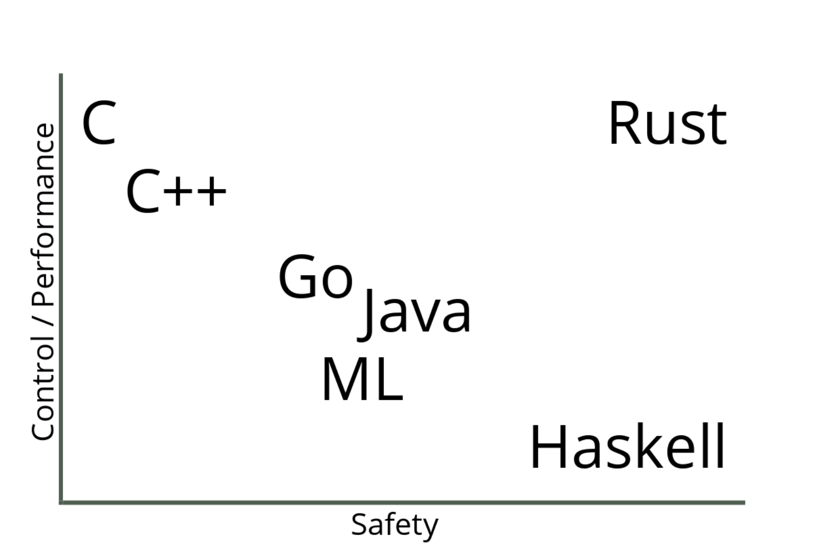


Figure 5: Rust vs other languages

What is Rust?

What does rust-lang.org say about Rust?

Performance

Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

Reliability

Rust’s rich type system and ownership model guarantee memory-safety and thread-safety — and enable you to eliminate many classes of bugs at compile-time.

Productivity

Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.

Use Cases

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

Things Written in Rust

- Servo/parts of Firefox
- Redox
- Ripgrep
- Dropbox's storage backend
- Many more. . .

The Language

Read the Rust Book
https://doc.rust-lang.org/book/
The Rust Playground
https://play.rust-lang.org/
Some examples from the rust book

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

```rust
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: &String) {
    println!("Hello {}!", name);
}

fn main() {
    let name = String::from("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

```rust
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;
```

Statements vs Expressions

Statement

```
let z = x + y;
```

Expression

```
{
    let z = x + y;
    z * y
}
```

Functions

```
fn hello() {
    println!("Hello");
}

fn add(x: i32, y: i32) -> i32 {
    x + y
}
```

Functions return the result of their last expression if it's not followed by a semi-colon

Structures

```
struct Person {
    name: String,
    age: u16,
}

let person = Person {
    name: String::from("Greg"),
    age: 32,
};
```

Methods

```rust
struct Point {
    x: i32,
    y: i32,
}

impl Point {
    fn new(x: i32, y: i32) -> Point {
        Point {
            x,
            y,
        }
    }
}
```

Methods cont.

```rust
impl Point {
    fn add(&mut self, other: &Point) {
        self.x += other.x;
        self.y += other.y;
    }
}

fn main() {
    let p1 = Point::new(1, 2);
    let p2 = Point::new(2, 3);
    // These are the same
    p1.add(&p2);
    Point::add(&mut p1, &p2);
}
```

Strings

Two types of strings

String slice

&str

```
let s1 = "Hello";
```

Owned string

String

```
let s1 = String::from("Hello");
let s2 = "World".to_owned();
let s3 = String::from("Foo").push_str("Bar");
```

Unit Type

() is the empty type

Functions that don’t specify a return type return ()

Enums

```
enum Direction {
    Left,
    Right,
}

enum IpAddr {
    V4(u8, u8, u8, u8),
    V6(String),
}
```

Matching

```rust
enum Coin {
    Penny,
    Nickel,
    Dime,
    Quarter,
}

fn value_in_cents(coin: Coin) -> u32 {
    match coin {
        Coin::Penny => 1,
        Coin::Nickel => 5,
        Coin::Dime => 10,
        Coin::Quarter => 25,
    }
}
```

Matching cont.

```rust
enum Coin {
    Penny,
    Nickel,
    Dime,
    Quarter,
}

fn is_a_penny(coin: Coin) -> bool {
    match coin {
        Coin::Penny => {
            println!("A penny!");
            true
        }
        _ => false,
    }
}
```

Matching cont. 2

```
enum IpAddr {
  V4(u8, u8, u8, u8),
  V6(String),
}

match ip_addr {
  IpAddr::V4(p1, p2, p3, p4) =>
    println!("{}.{}.{}.{}.", p1, p2, p3, p4),
  IpAddr::V6(s) => println!("{}", s),
}
```

If-let

```
match ip_addr {
    IpAddr::V6(s) => println!("{}", s),
    _ => (),
}
```

```
if let IpAddr::V6(s) = ip_addr {
    println!("{}", s);
}
```

Panic

```rust
fn main() {
    panic!("crash and burn");
}
```

```rust
fn main() {
    let v = vec![1, 2, 3];

    v[99];
}
```

Result

```
enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

Result Ex.

```
use std::fs::File;

fn main() {
    let f = File::open("hello.txt");

    let f = match f {
        Ok(file) => file,
        Err(error) => {
            panic!("There was a problem opening the file: 
        },
    };
}
```

Result Ex. cont.

```
use std::fs::File;

fn main() {
    let f = File::open("hello.txt").unwrap();
}
```

Propagate Errors

```rust
use std::io;
use std::io::Read;
use std::fs::File;

fn read_username_from_file() -> Result<String, io::Error>
    let f = File::open("hello.txt");

    let mut f = match f {
        Ok(file) => file,
        Err(e) => return Err(e),
    };

    let mut s = String::new();

    match f.read_to_string(&mut s) {
        Ok(_) => Ok(s),
        Err(e) => Err(e),
    }
```

Propagate Errors cont.

```
use std::io;
use std::io::Read;
use std::fs::File;

fn read_username_from_file() -> Result<String, io::Error> {
    let mut f = File::open("hello.txt")?;
    let mut s = String::new();
    f.read_to_string(&mut s)?;
    Ok(s)
}
```

Option

```
enum Option<T> {
    Some(T),
    None,
}
```

Generics

```rust
struct Point<T> {
    x: T,
    y: T,
}

impl<T> Point<T> {
    fn x(&self) -> &T {
        &self.x
    }
}

fn main() {
    let p = Point { x: 5, y: 10 };

    println!("p.x = {}", p.x());
}
```

Traits

```
trait MakeSound {
    fn make_sound() -> String;
}

struct Dog;

impl MakeSound for Dog {
    fn make_sound() -> String {
        String::from("bark!")
    }
}
```

Traits in Generics

```
fn are_equal<T: Eq>(x: T, y: T) -> bool {
    x == y
}
```

Borrow Checker

```
{
    let r;                // ---------+-- 'a
                          //          |
    {                     //          |
        let x = 5;        // -+-- 'b  |
        r = &x;           //  |       |
    }                     // -+       |
                          //          |
    println!("r: {}", r); //          |
}
```

Borrow Checker cont.

```
{
    let x = 5;            // ----------+-- 'b
                          //           |
    let r = &x;           // --+-- 'a  |
                          //   |       |
    println!("r: {}", r); //   |       |
                          // --+       |
}
```

Lifetimes

```
fn longest(x: &str, y: &str) -> &str {
    if x.len() > y.len() {
        x
    } else {
        y
    }
}

fn longest<'a>(x: &'a str, y: &'a str) -> &'a str {
    if x.len() > y.len() {
        x
    } else {
        y
    }
}
```

Lifetimes cont.

```
fn main() {
    let string1 = String::from("long string is long");

    {
        let string2 = String::from("xyz");
        let result = longest(string1.as_str(), string2.as_s
        println!("The longest string is {}", result);
    }
}
```

Lifetimes cont. 2

```
fn main() {
    let string1 = String::from("long string is long");
    let result;
    {
        let string2 = String::from("xyz");
        result = longest(string1.as_str(), string2.as_str()
    }
    println!("The longest string is {}", result);
}
```

Modules

```
pub mod sound {
    pub mod instrument {
        pub fn clarinet() {
            // Function body code goes here
        }
    }
}

fn main() {
    // Absolute path
    crate::sound::instrument::clarinet();

    // Relative path
    sound::instrument::clarinet();
}
```

Modules cont.

```
pub struct Point {
    pub x: i32,
    pub y: i32,
}
```

Syncronazation

The borrow checker also prevent shared mutablity between thread and prevents data races

Rust also provides safe and effective syncronazation primatives

Cargo

The best build tool

- Build all of your code with out of the box
- Pull in all dependencies with no headaches
- It just works!

Resources

- The Rust website: `https://www.rust-lang.org/`
- The Rust book: `https://doc.rust-lang.org/book/`
- The Rust playground `https://play.rust-lang.org/`
- Rust by example: `https://github.com/rust-lang/rust-by-example`