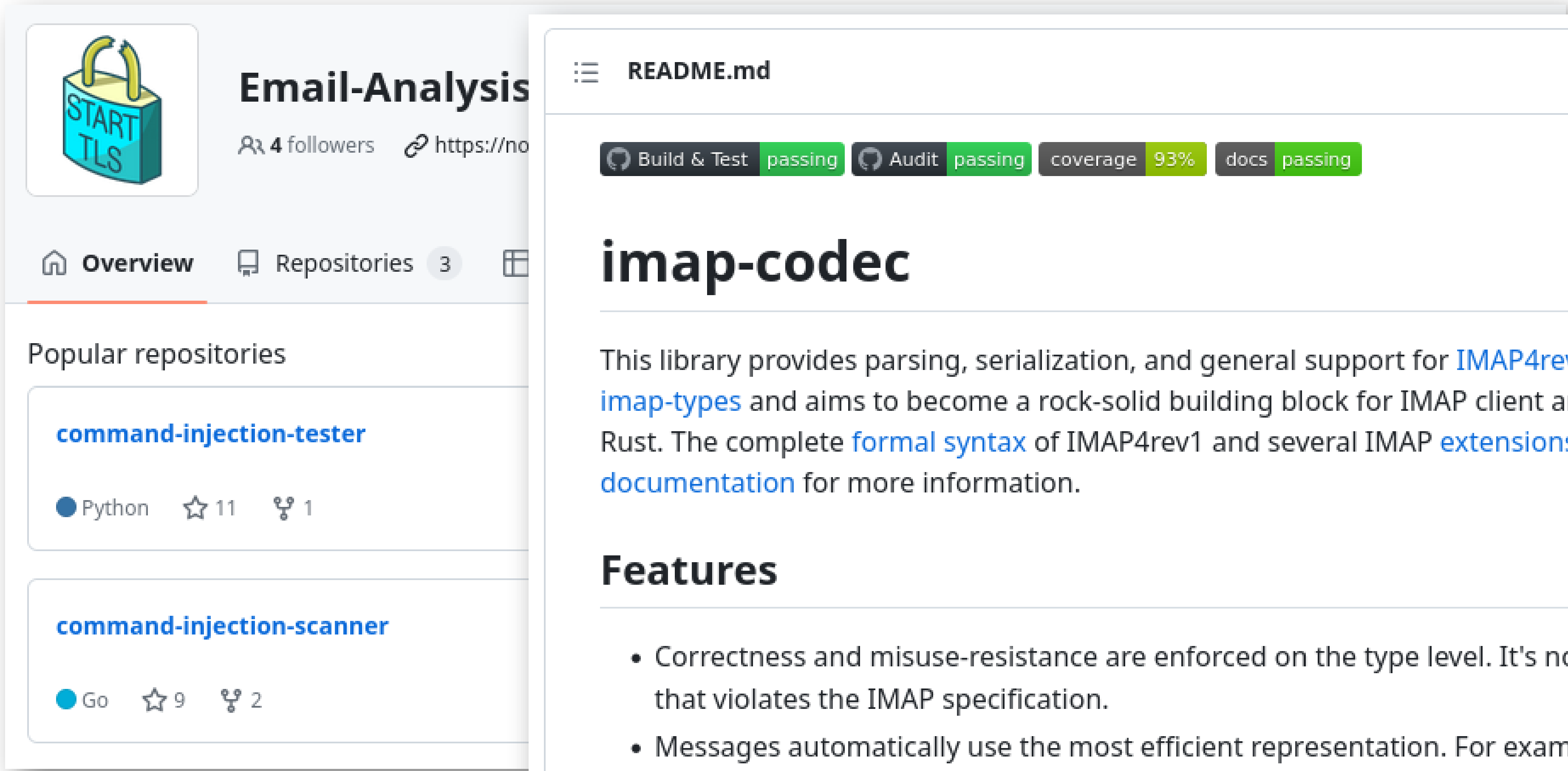
Rust: Eine moderne Alternative zu C und C++

Damian Poddebniak

OWASP Hamburg Stammtisch, 06.07.2023

#rust-hamburg:matrix.org



Email-Analysis

4 followers https://no

Overview Repositories 3

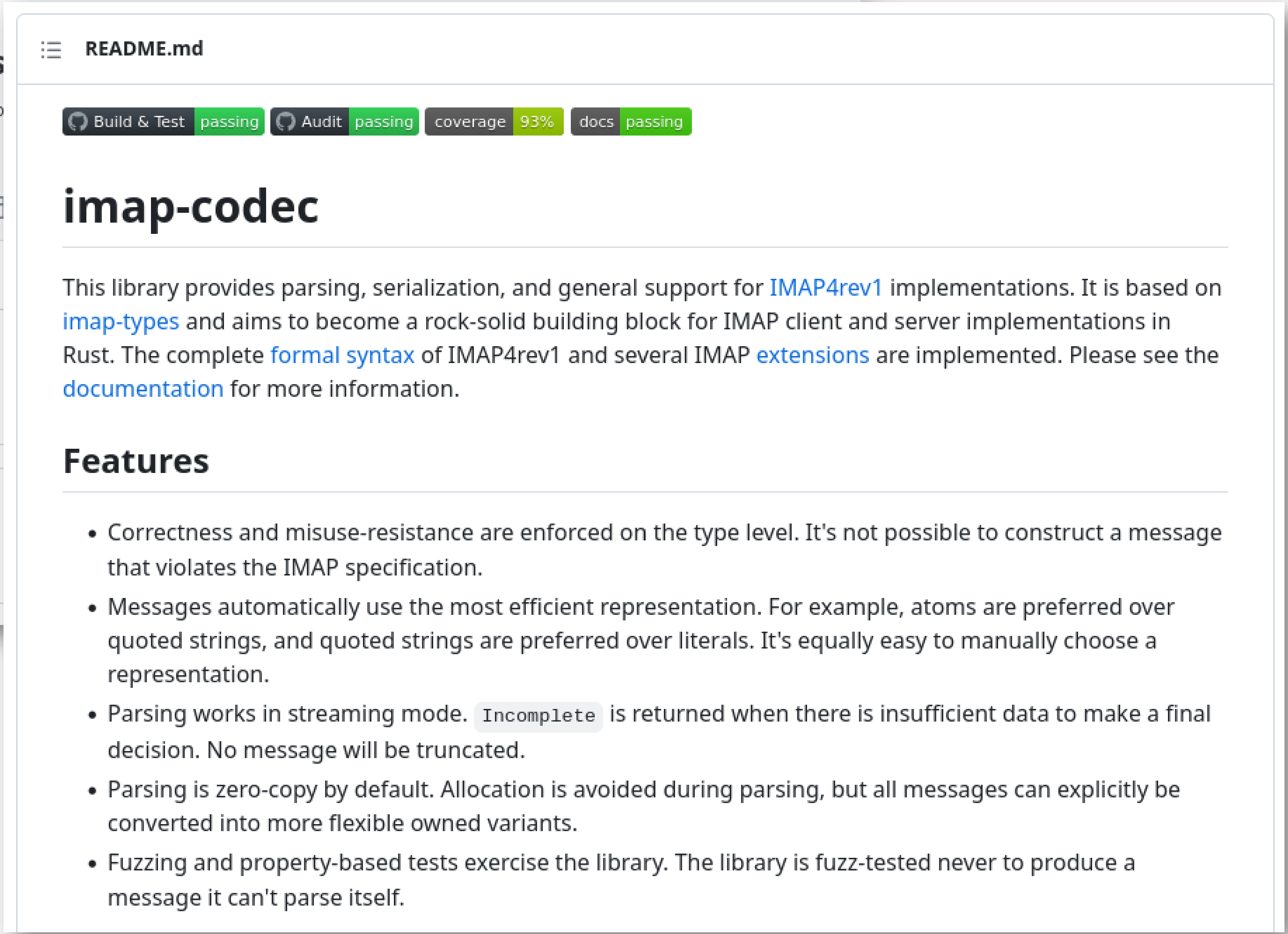
Popular repositories

command-injection-tester

Python 11 1

command-injection-scanner

Go 9 2



README.md

Build & Test passing | Audit passing | coverage 93% | docs passing

imap-codec

This library provides parsing, serialization, and general support for IMAP4rev1 implementations. It is based on imap-types and aims to become a rock-solid building block for IMAP client and server implementations in Rust. The complete formal syntax of IMAP4rev1 and several IMAP extensions are implemented. Please see the documentation for more information.

Features

- Correctness and misuse-resistance are enforced on the type level. It's not possible to construct a message that violates the IMAP specification.
- Messages automatically use the most efficient representation. For example, atoms are preferred over quoted strings, and quoted strings are preferred over literals. It's equally easy to manually choose a representation.
- Parsing works in streaming mode. `Incomplete` is returned when there is insufficient data to make a final decision. No message will be truncated.
- Parsing is zero-copy by default. Allocation is avoided during parsing, but all messages can explicitly be converted into more flexible owned variants.
- Fuzzing and property-based tests exercise the library. The library is fuzz-tested never to produce a message it can't parse itself.

Agenda

Teil 1 – Einleitung / Motivation

Teil 2 – Live Coding

- Implementierung einer HTTP Library
  - Tooling
    - Tests
    - Dokumentation
    - Debugging
  - Features
    - Ownership & Borrowing
    - Algebraische Datentypen

Eckdaten

- Entstanden bei Mozilla Research
- Stabil
- Kompiliert (LLVM)
  - High-performance
  - Platform support
- Features
  - Ownership & Borrowing (Speichersicherheit)
  - Algebraische Datentypen
  - ...
- Community-Projekt

target	notes
aarch64-unknown-linux-gnu	ARM64 Linux (kernel 4.1, glibc 2.17+) 1
i	
i	
i	
x	
x	
x	
x	

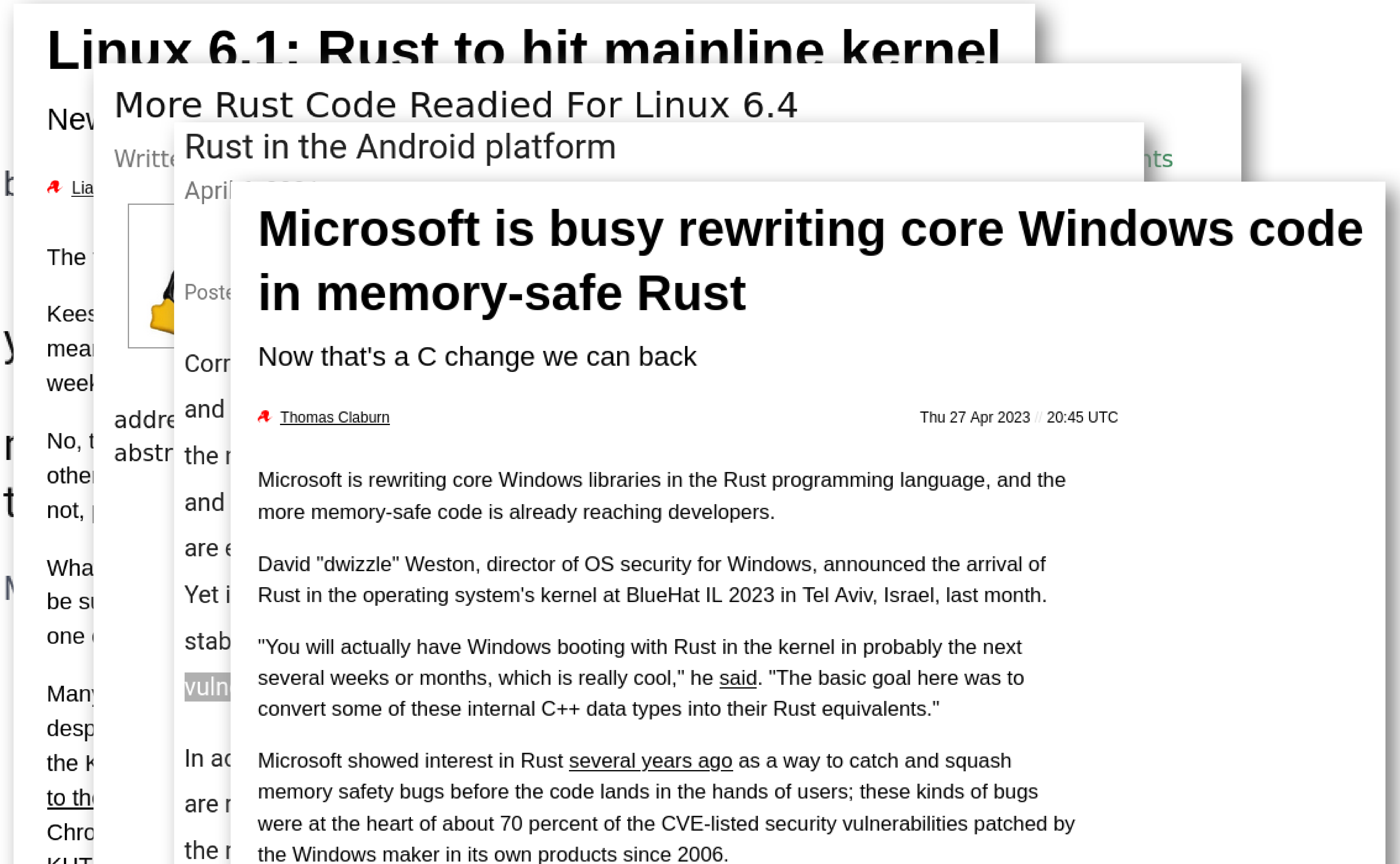
Governance

How Rust is built by its community

RFC process

Each major decision in Rust starts as a Request for Comments (RFC). Everyo
discuss the proposal, to work toward a shared understanding of the tradeof
sometimes arduous, this community deliberation is Rust's secret sauce for

Teams



Linux 6.1: Rust to hit mainline kernel

More Rust Code Readied For Linux 6.4

Rust in the Android platform

Microsoft is busy rewriting core Windows code in memory-safe Rust

Now that's a C change we can back

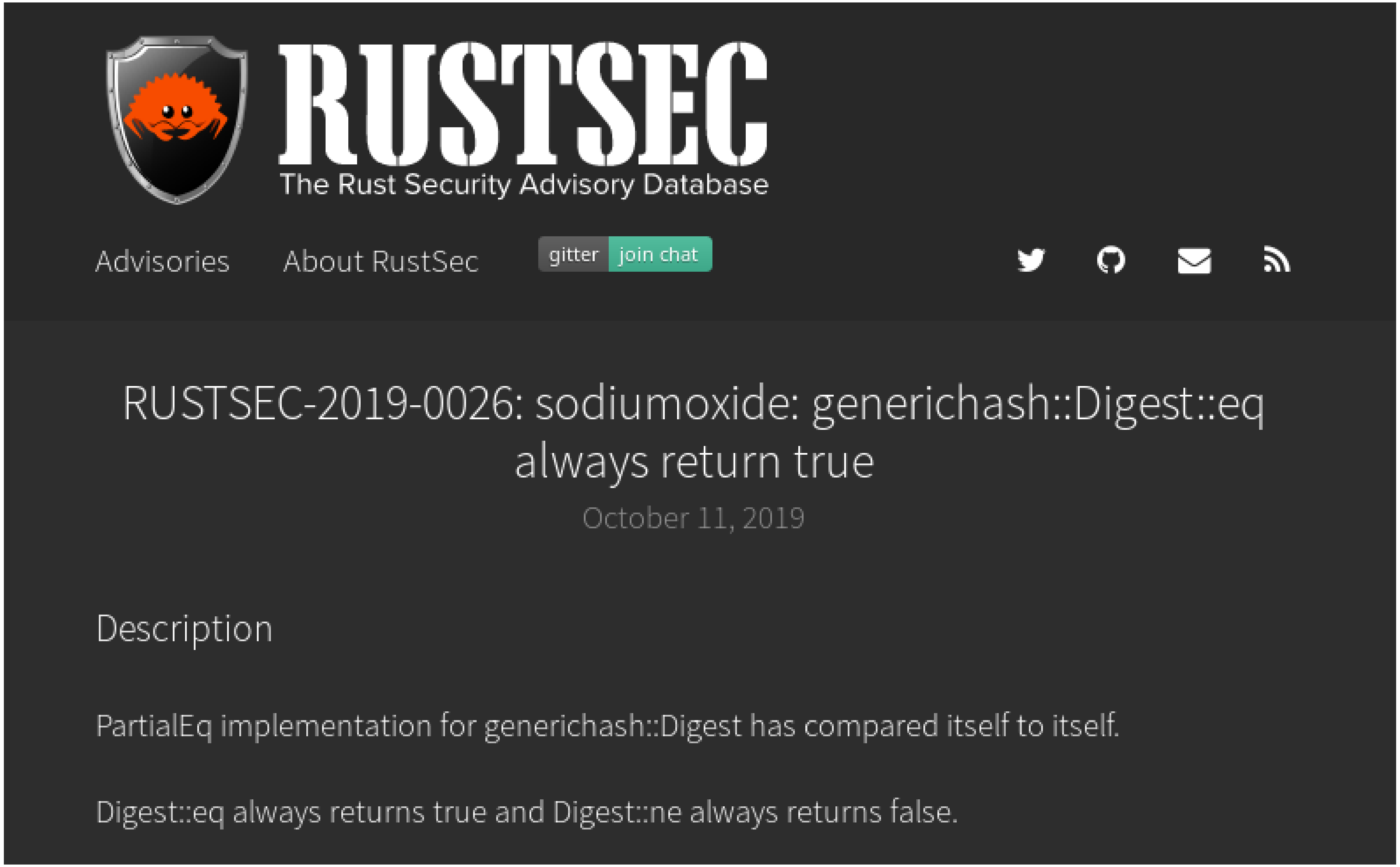
Thomas Claburn Thu 27 Apr 2023 // 20:45 UTC

Microsoft is rewriting core Windows libraries in the Rust programming language, and the more memory-safe code is already reaching developers.

David "dwizzle" Weston, director of OS security for Windows, announced the arrival of Rust in the operating system's kernel at BlueHat IL 2023 in Tel Aviv, Israel, last month.

"You will actually have Windows booting with Rust in the kernel in probably the next several weeks or months, which is really cool," he said. "The basic goal here was to convert some of these internal C++ data types into their Rust equivalents."

Microsoft showed interest in Rust several years ago as a way to catch and squash memory safety bugs before the code lands in the hands of users; these kinds of bugs were at the heart of about 70 percent of the CVE-listed security vulnerabilities patched by the Windows maker in its own products since 2006.



RUSTSEC-2019-0026: sodiumoxide: generichash::Digest::eq always return true

October 11, 2019

Description

PartialEq implementation for generichash::Digest has compared itself to itself.

Digest::eq always returns true and Digest::ne always returns false.

Memory Safety (Intuition)

Spatial

- {Stack,Heap}-based buffer overflows
- Out-of-bounds {read,write}

Temporal

- Using uninitialized memory
- Use-after-free
- Double-free
- NULL-pointer dereference

Memory Safety (Intuition)

Spatial

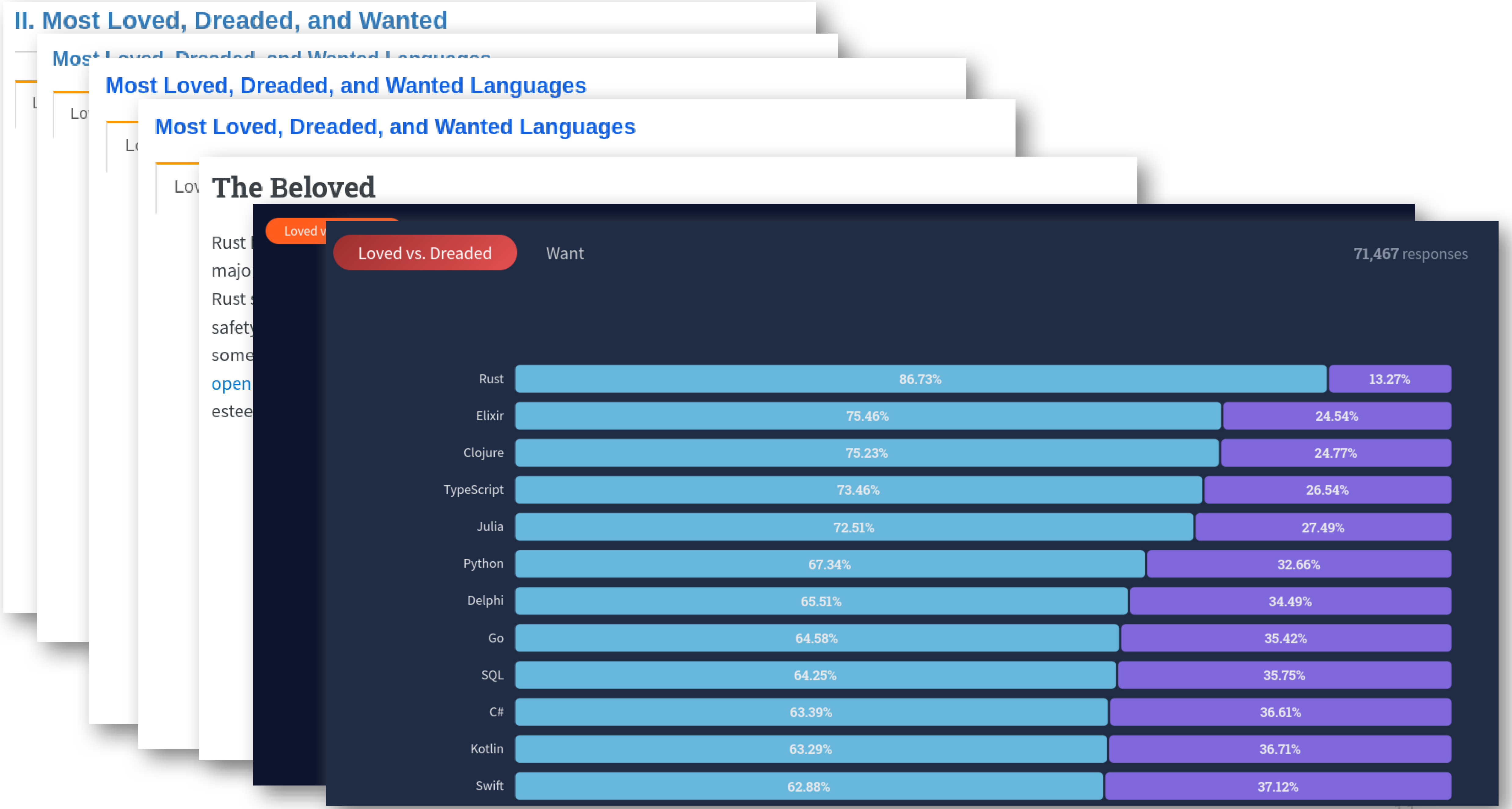
- ~~{Stack,Heap}-based buffer overflows~~ --> bounds checking
- ~~Out-of-bounds {read,write}~~ --> bounds checking

Temporal

- ~~Using uninitialized memory~~ --> not allowed
- ~~Use-after-free~~ --> Ownership
- ~~Double-free~~ --> Ownership
- ~~NULL-pointer dereference~~ --> Option<T>

Rank	ID	Name	Score	KEV Count (CVEs)	Rank Change vs. 2021
1	CWE-787	Out-of-bounds Write	64.20	62	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.97	2	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	22.11	7	+3 ▲
4	CWE-20	Improper Input Validation	20.63	20	0
5	CWE-125	Out-of-bounds Read	17.67	1	-2 ▼
6	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	17.53	32	-1 ▼
7	CWE-416	Use After Free	15.50	28	0
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.08	19	0
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.53	1	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	9.56	6	0
11	CWE-476	NULL Pointer Dereference	7.15	0	+4 ▲
12	CWE-502	Deserialization of Untrusted Data	6.68	7	+1 ▲
13	CWE-190	Integer Overflow or Wraparound	6.53	2	-1 ▼
14	CWE-287	Improper Authentication	6.35	4	0
15	CWE-798	Use of Hard-coded Credentials	5.66	0	+1 ▲
16	CWE-862	Missing Authorization	5.53	1	+2 ▲
17	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	5.42	5	+8 ▲
18	CWE-306	Missing Authentication for Critical Function	5.15	6	-7 ▼
19	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	4.85	6	-2 ▼
20	CWE-276	Incorrect Default Permissions	4.84	0	-1 ▼
21	CWE-918	Server-Side Request Forgery (SSRF)	4.27	8	+3 ▲
22	CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	3.57	6	+11 ▲
23	CWE-400	Uncontrolled Resource Consumption	3.56	2	+4 ▲
24	CWE-611	Improper Restriction of XML External Entity Reference	3.38	0	-1 ▼
25	CWE-94	Improper Control of Generation of Code ('Code Injection')	3.32	4	+3 ▲

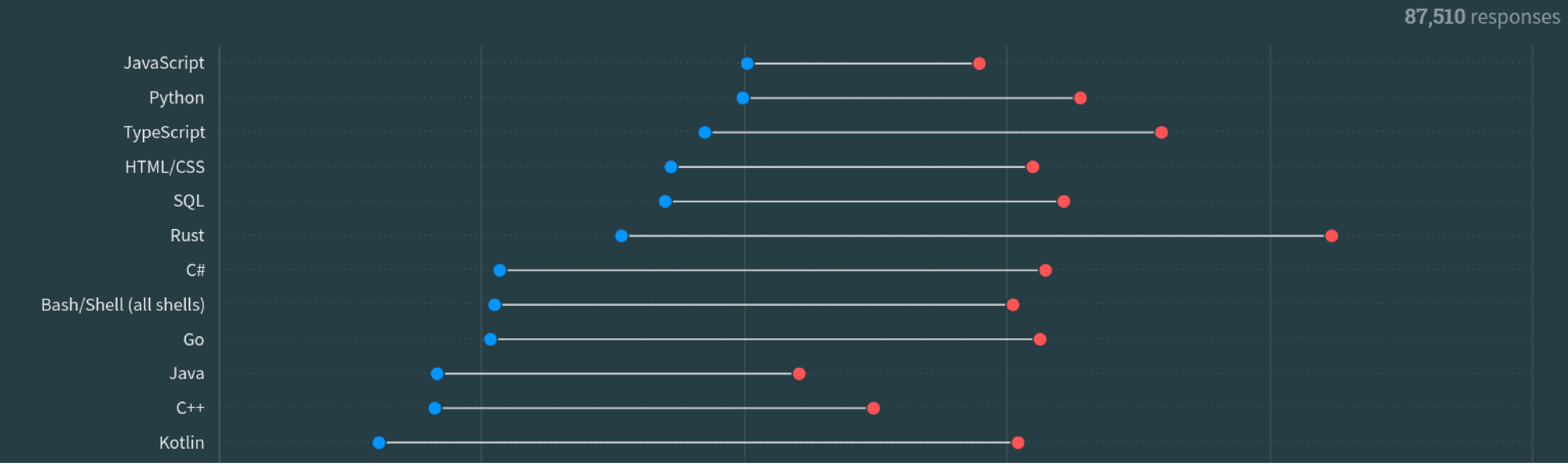
Rank	ID	Name	Score	CVEs in KEV	Rank Change vs. 2022
1	CWE-787	Out-of-bounds Write	63.72	70	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.54	4	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	34.27	6	0
4	CWE-416	Use After Free	16.71	44	+3
5	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	15.65	23	+1
6	CWE-20	Improper Input Validation	15.50	35	-2
7	CWE-125	Out-of-bounds Read	14.60	2	-2
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.11	16	0
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.73	0	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	10.41	5	0
11	CWE-862	Missing Authorization	6.90	0	+5
12	CWE-476	NULL Pointer Dereference	6.59	0	-1
13	CWE-287	Improper Authentication	6.39	10	+1
14	CWE-190	Integer Overflow or Wraparound	5.89	4	-1
15	CWE-502	Deserialization of Untrusted Data	5.56	14	-3
16	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	4.95	4	+1
17	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	4.75	7	+2
18	CWE-798	Use of Hard-coded Credentials	4.57	2	-3
19	CWE-918	Server-Side Request Forgery (SSRF)	4.56	16	+2
20	CWE-306	Missing Authentication for Critical Function	3.78	8	-2
21	CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	3.53	8	+1
22	CWE-269	Improper Privilege Management	3.31	5	+7
23	CWE-94	Improper Control of Generation of Code ('Code Injection')	3.30	6	+2
24	CWE-863	Incorrect Authorization	3.16	0	+4
25	CWE-276	Incorrect Default Permissions	3.16	0	-5

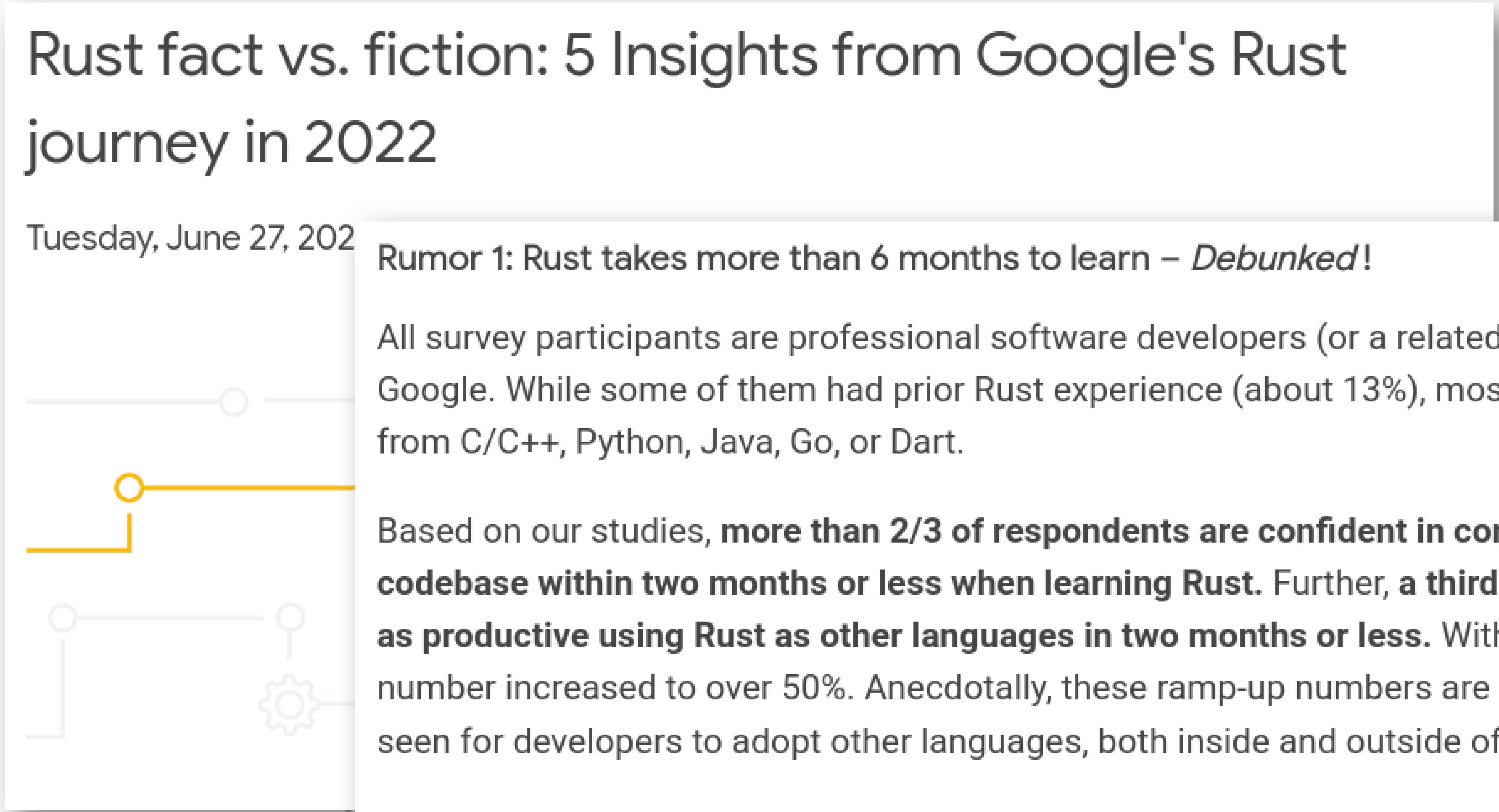


II. Most Loved, Dreaded, and Wanted
Most Loved, Dreaded, and Wanted Languages
Most Loved, Dreaded, and Wanted Languages
Most Loved, Dreaded, and Wanted Languages
The Beloved
Rust
majo
Rust
safety
some
open
estee
Loved v
Loved vs. Dreaded
Want
71,467 responses
Rust 86.73% 13.27%
Elixir 75.46% 24.54%
Clojure 75.23% 24.77%
TypeScript 73.46% 26.54%
Julia 72.51% 27.49%
Python 67.34% 32.66%
Delphi 65.51% 34.49%
Go 64.58% 35.42%
SQL 64.25% 35.75%
C# 63.39% 36.61%
Kotlin 63.29% 36.71%
Swift 62.88% 37.12%

Programming, scripting, and markup languages

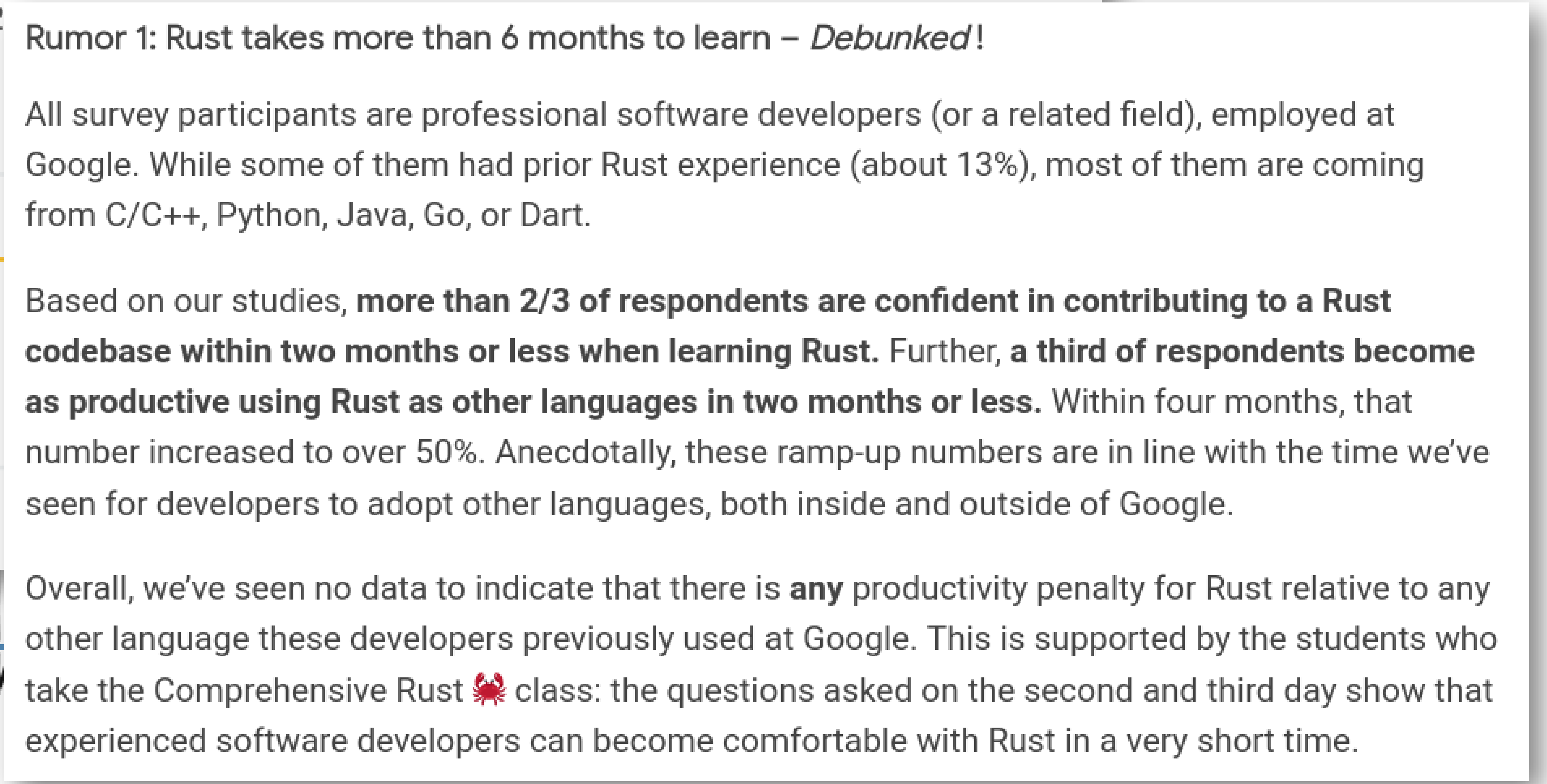
Rust is the most admired language, more than 80% of developers that use it want to use it again next year. Compare this to the least admired language: MATLAB. Less than 20% of developers who used this language want to use it again next year.





Rust fact vs. fiction: 5 Insights from Google's Rust journey in 2022

Tuesday, June 27, 202



Rumor 1: Rust takes more than 6 months to learn – *Debunked*!

All survey participants are professional software developers (or a related field), employed at Google. While some of them had prior Rust experience (about 13%), most of them are coming from C/C++, Python, Java, Go, or Dart.

Based on our studies, **more than 2/3 of respondents are confident in contributing to a Rust codebase within two months or less when learning Rust.** Further, **a third of respondents become as productive using Rust as other languages in two months or less.** Within four months, that number increased to over 50%. Anecdotally, these ramp-up numbers are in line with the time we've seen for developers to adopt other languages, both inside and outside of Google.

Overall, we've seen no data to indicate that there is **any** productivity penalty for Rust relative to any other language these developers previously used at Google. This is supported by the students who take the Comprehensive Rust 🦀 class: the questions asked on the second and third day show that experienced software developers can become comfortable with Rust in a very short time.

Alternativen

Überblick

Sichere C Dialekte

- Cyclone, Deputy, MISRA-C, ...

Sichere C Implementierungen

- MSVC RTC Compiler, Fail-Safe C, Ccured, FORTIFY_SOURCE, …

Statische Code Analyse

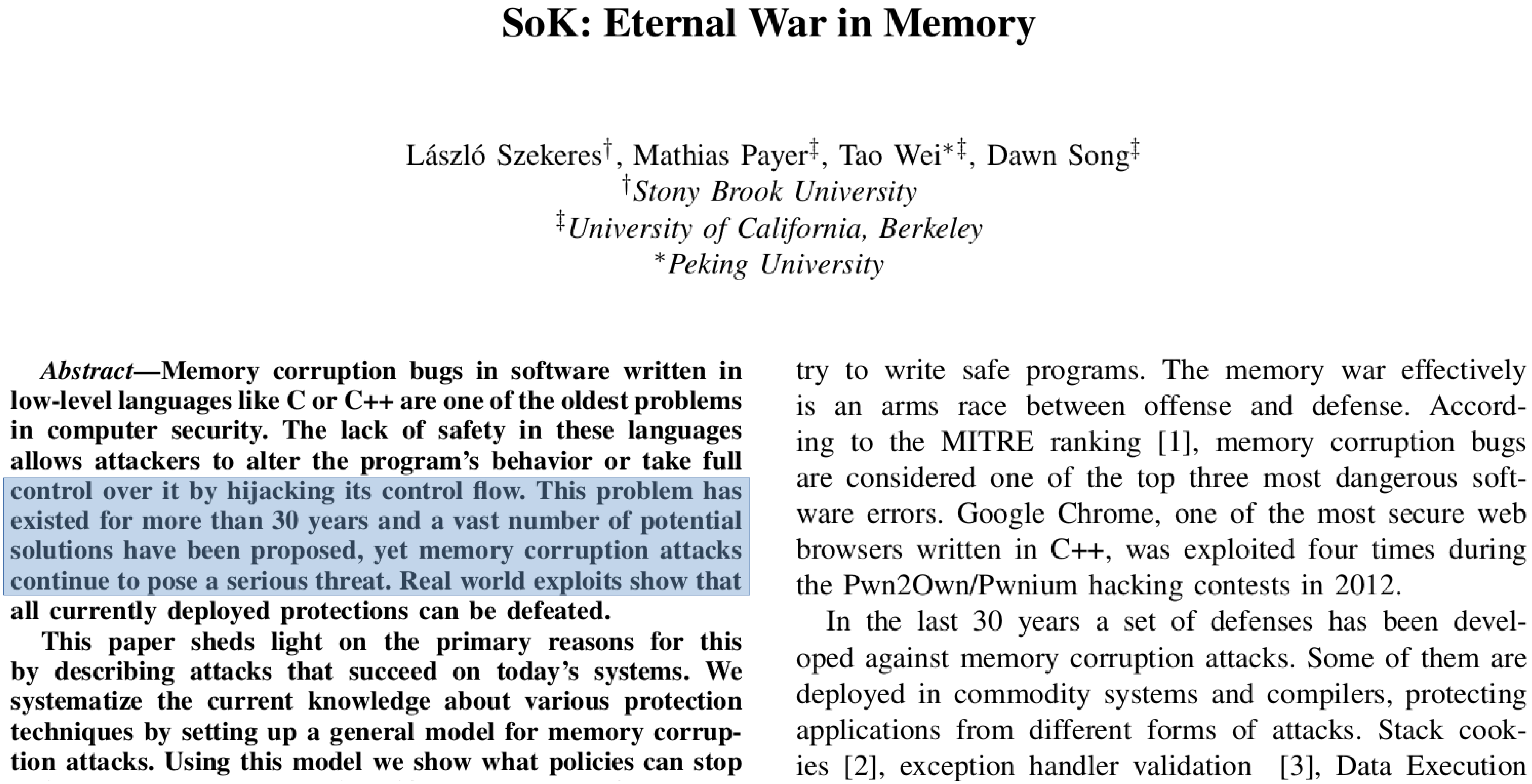
- CodeSonar, Coverity Static Analysis, ...

Testing

- Asan, AFL, libFuzzer, …

Mitigations

- DEP, Stack Canaries, ASLR, CFI, ...



SoK: Eternal War in Memory

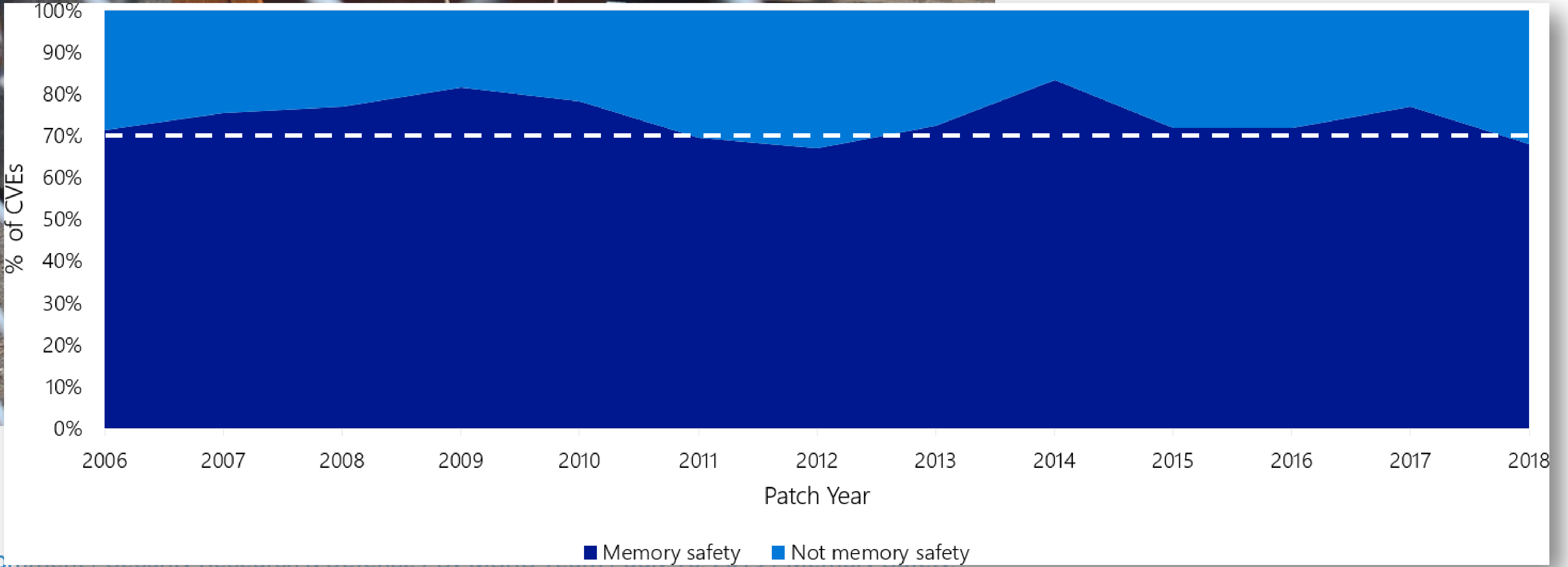
László Szekeres[†], Mathias Payer[‡], Tao Wei[*][‡], Dawn Song[‡]
[†]*Stony Brook University*
[‡]*University of California, Berkeley*
[*]*Peking University*

***Abstract*—Memory corruption bugs in software written in low-level languages like C or C++ are one of the oldest problems in computer security. The lack of safety in these languages allows attackers to alter the program's behavior or take full control over it by hijacking its control flow. This problem has existed for more than 30 years and a vast number of potential solutions have been proposed, yet memory corruption attacks continue to pose a serious threat. Real world exploits show that all currently deployed protections can be defeated.**

This paper sheds light on the primary reasons for this by describing attacks that succeed on today's systems. We systematize the current knowledge about various protection techniques by setting up a general model for memory corruption attacks. Using this model we show what policies can stop

try to write safe programs. The memory war effectively is an arms race between offense and defense. According to the MITRE ranking [1], memory corruption bugs are considered one of the top three most dangerous software errors. Google Chrome, one of the most secure web browsers written in C++, was exploited four times during the Pwn2Own/Pwnium hacking contests in 2012.

In the last 30 years a set of defenses has been developed against memory corruption attacks. Some of them are deployed in commodity systems and compilers, protecting applications from different forms of attacks. Stack cookies [2], exception handler validation [3], Data Execution

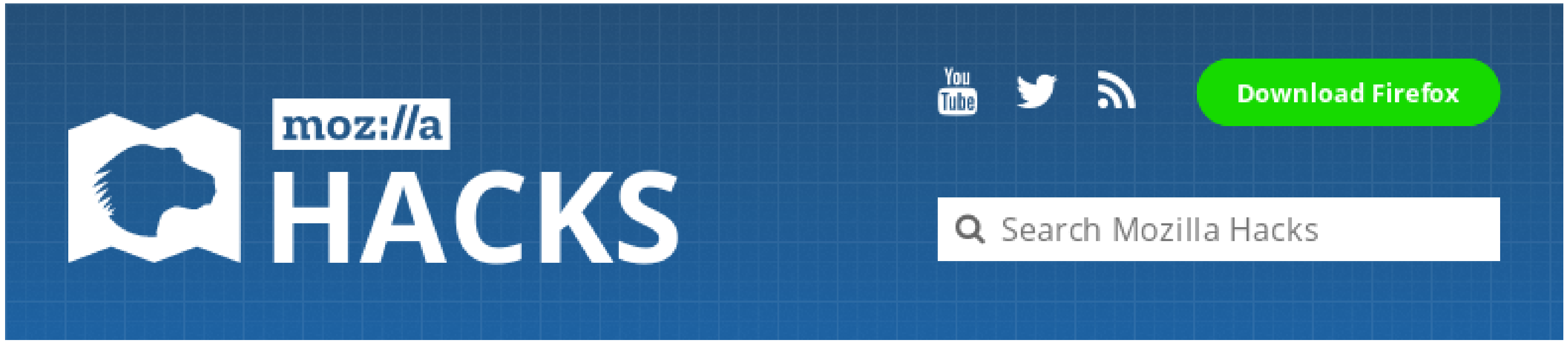


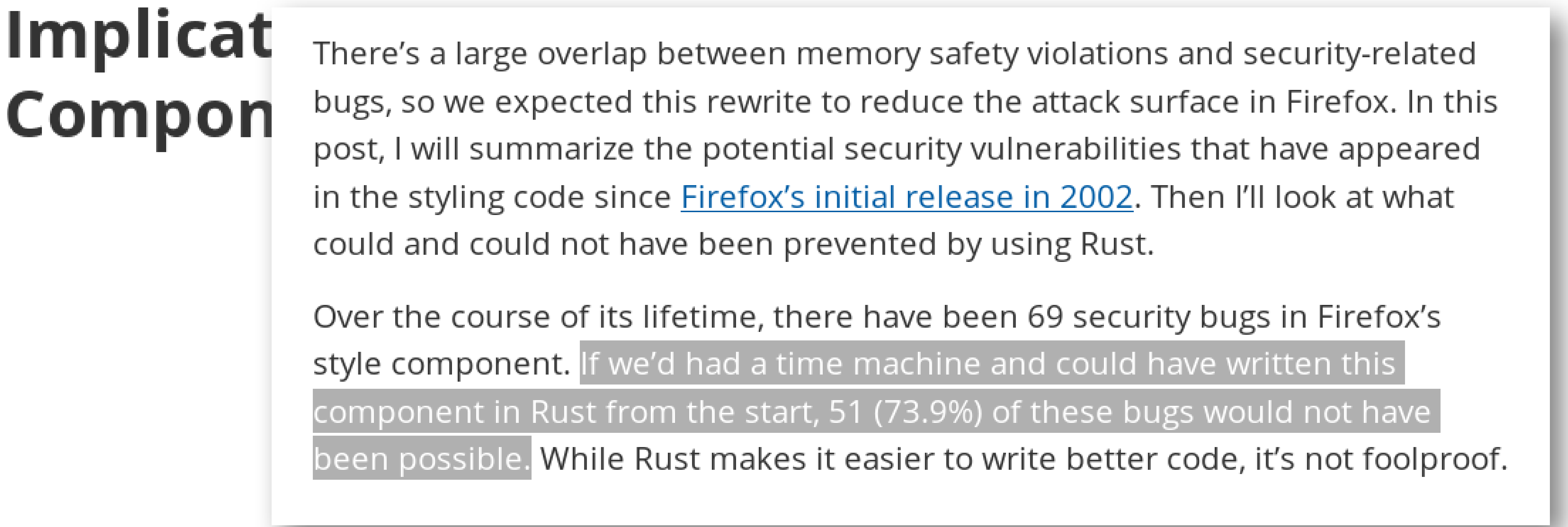
A

1 Cc

Rust, Safe Systems Programming Languages, Secure Development

What if we could eliminate an entire class of vulnerabilities before they ever happened?

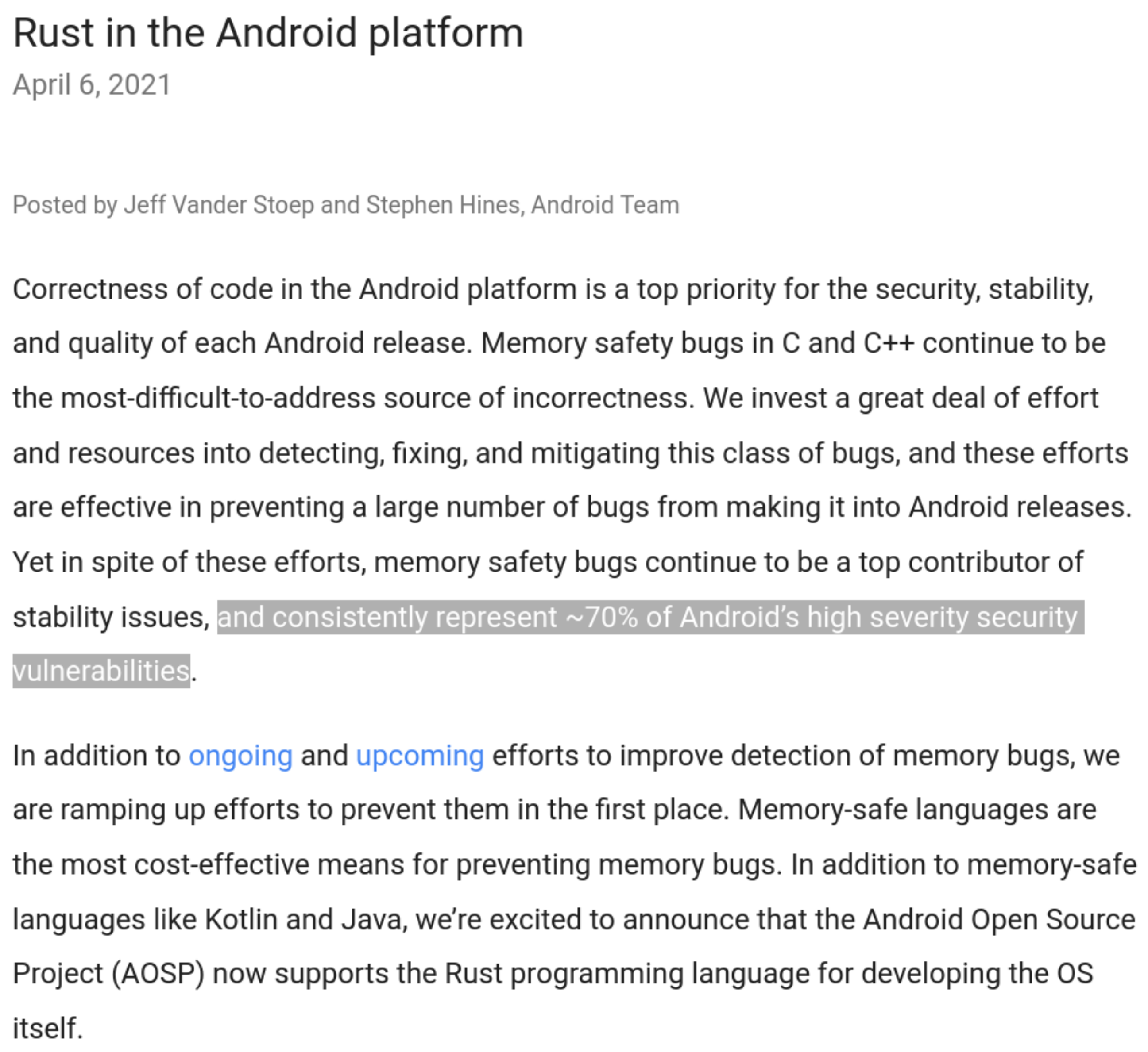




Implicat
Compon

There’s a large overlap between memory safety violations and security-related bugs, so we expected this rewrite to reduce the attack surface in Firefox. In this post, I will summarize the potential security vulnerabilities that have appeared in the styling code since Firefox’s initial release in 2002. Then I’ll look at what could and could not have been prevented by using Rust.

Over the course of its lifetime, there have been 69 security bugs in Firefox’s style component. If we’d had a time machine and could have written this component in Rust from the start, 51 (73.9%) of these bugs would not have been possible. While Rust makes it easier to write better code, it’s not foolproof.



Rust in the Android platform

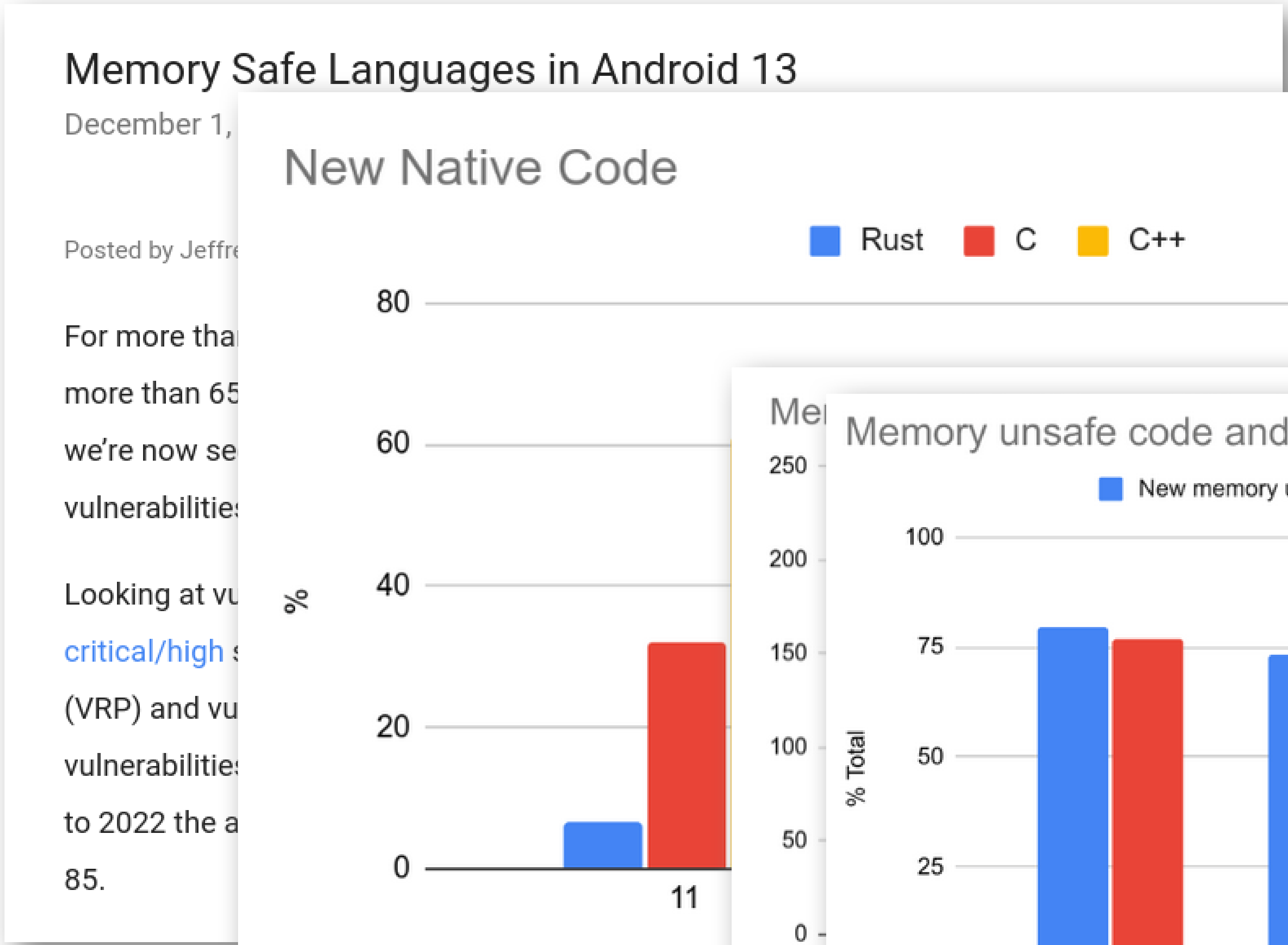
April 6, 2021

Posted by Jeff Vander Stoep and Stephen Hines, Android Team

Correctness of code in the Android platform is a top priority for the security, stability, and quality of each Android release. Memory safety bugs in C and C++ continue to be the most-difficult-to-address source of incorrectness. We invest a great deal of effort and resources into detecting, fixing, and mitigating this class of bugs, and these efforts are effective in preventing a large number of bugs from making it into Android releases. Yet in spite of these efforts, memory safety bugs continue to be a top contributor of stability issues, and consistently represent ~70% of Android's high severity security vulnerabilities.

In addition to ongoing and upcoming efforts to improve detection of memory bugs, we are ramping up efforts to prevent them in the first place. Memory-safe languages are the most cost-effective means for preventing memory bugs. In addition to memory-safe languages like Kotlin and Java, we're excited to announce that the Android Open Source Project (AOSP) now supports the Rust programming language for developing the OS itself.

https://security.googleblog.com/2021/04/rust-in-android-platform.html



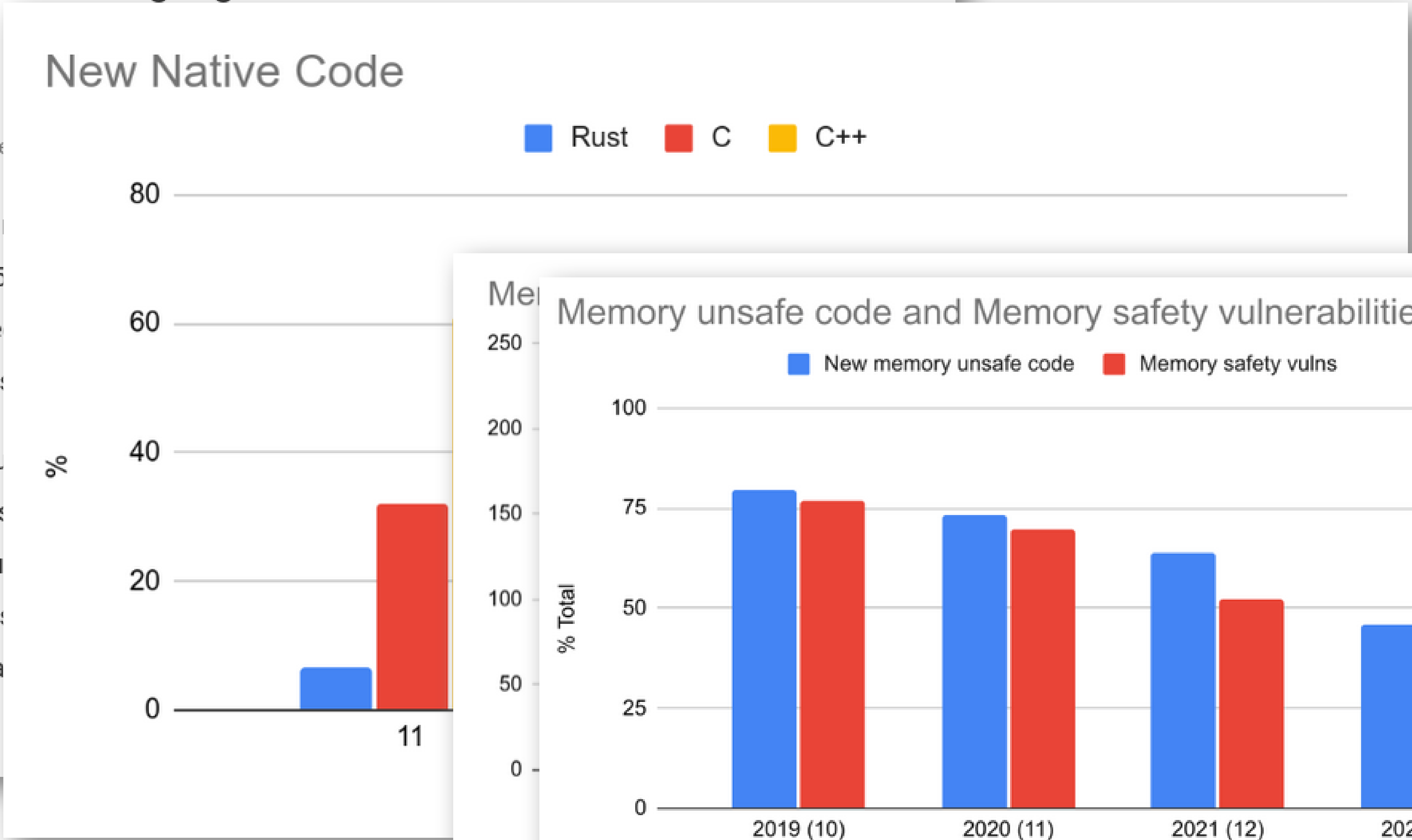
Memory Safe Languages in Android 13

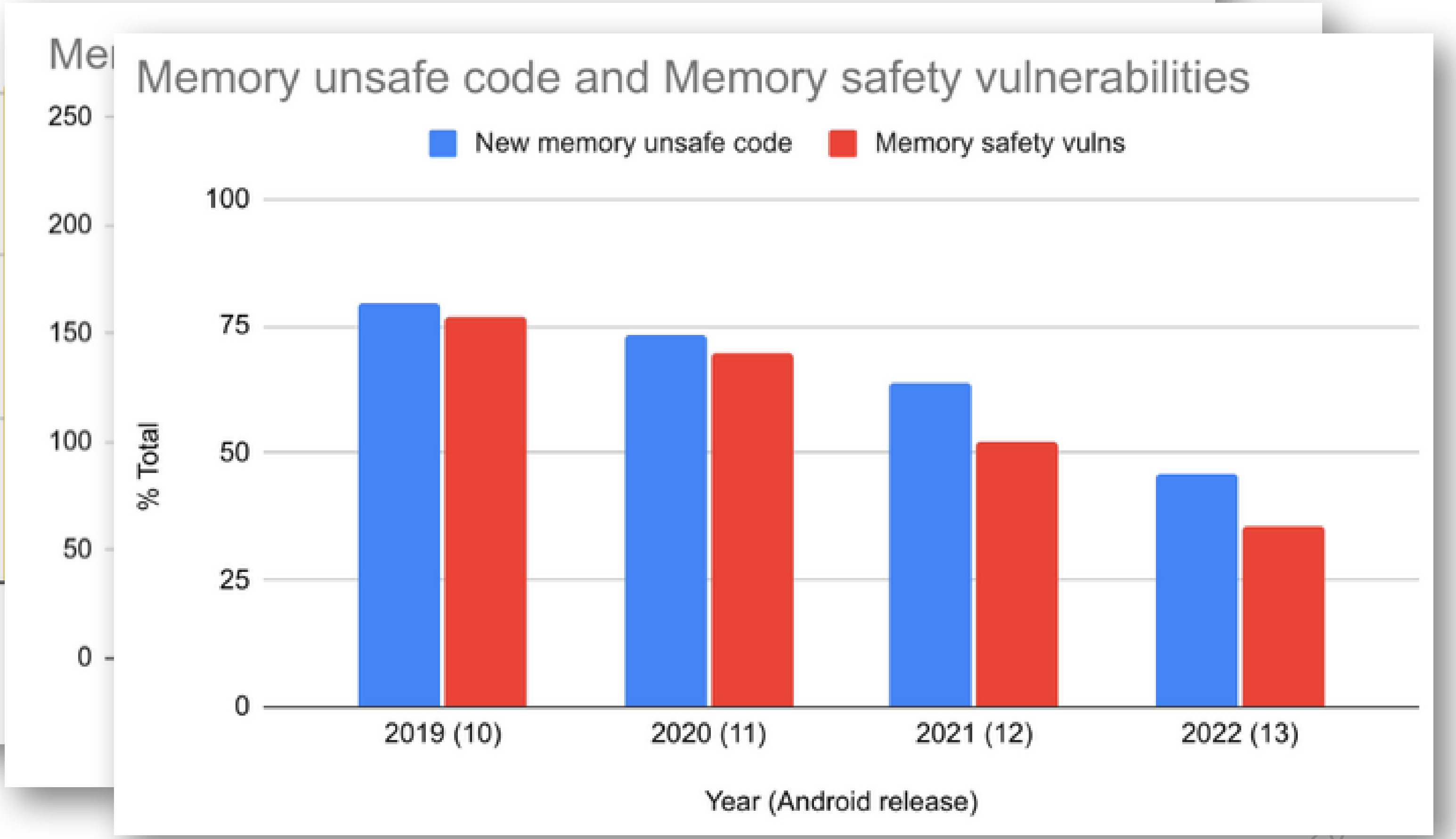
December 1,

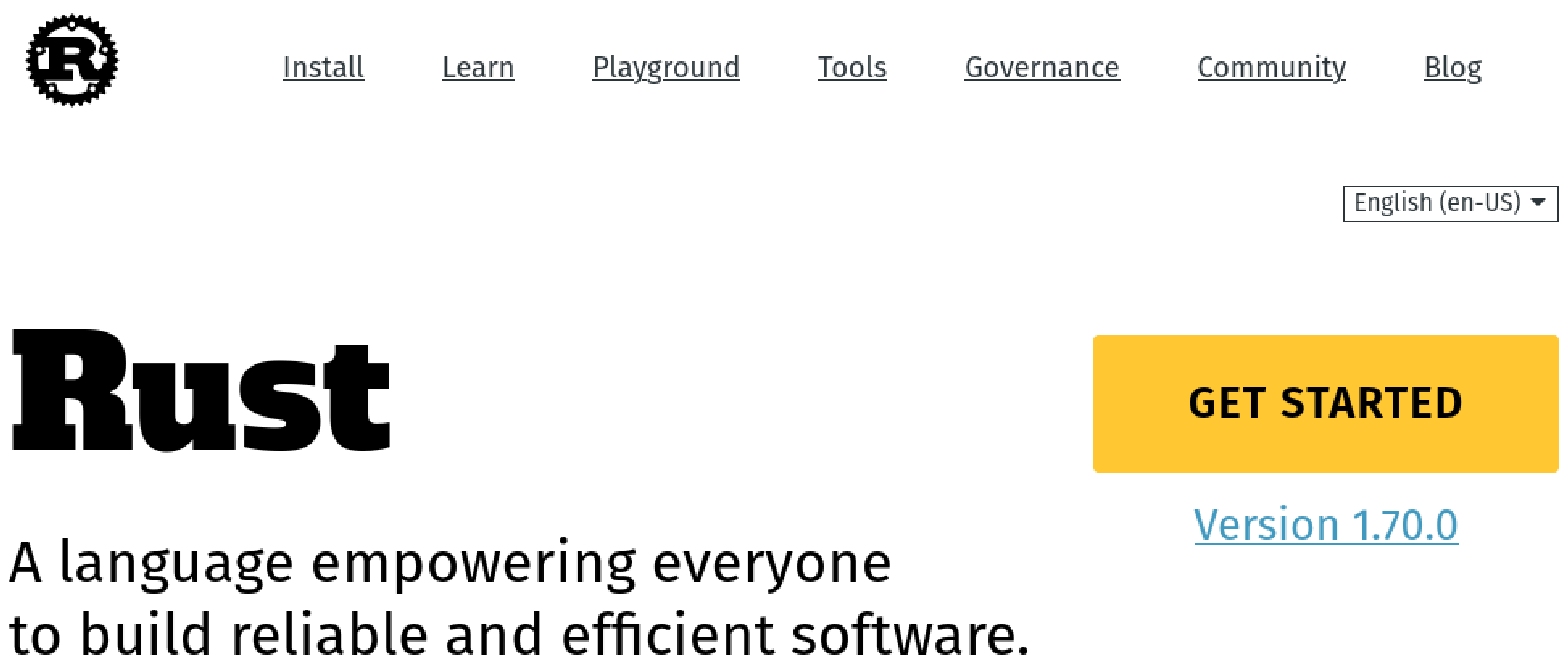
Posted by Jeffre

For more tha
more than 65
we're now se
vulnerabilitie

Looking at vu
critical/high s
(VRP) and vu
vulnerabilitie
to 2022 the a
85.







Rust

A language empowering everyone
to build reliable and efficient software.

Rust: Eine moderne Alternative zu C und C++

Damian Poddebniak

OWASP Hamburg Stammtisch, 06.07.2023

#rust-hamburg:matrix.org

BACKUP

Messaging Layer Security

```
struct {
  ProposalOrRefType type;
  select (ProposalOrRef.type) {
    case proposal:  Proposal proposal;
    case reference: ProposalRef reference;
  };
} ProposalOrRef;
```

```
struct {
  ProposalType proposal_type;
  select (Proposal.proposal_type) {
    case add:    Add;
    case update: Update;
    case remove: Remove;
    case psk:    PreSharedKey;
    ...
  };
} Proposal;
```

Rust Code

```
enum ProposalOrRef {
  Proposal(Proposal),
  ProposalRef(ProposalRef),
}
```

```
enum Proposal {
  Add(Add),
  Update(Update),
  Remove(Remove),
  Psk(Psk),
  ...
}
```