CIKLUM | SPEAKER'S CORNER

April, 16 | 17:00 (CET) | English

Why can't we afford to ignore Rust?

Software security in numbers

Adrian Panicek

Senior Embedded Software Engineer at Ciklum

Experiences of tomorrow. engineered together.

CIKLUM

We transform how people experience the business. all through next generation technology.

What we do:

- Product Engineering
- Intelligent automation
- Data & analytics

2002
founded

4000+
professionals

offices

offices

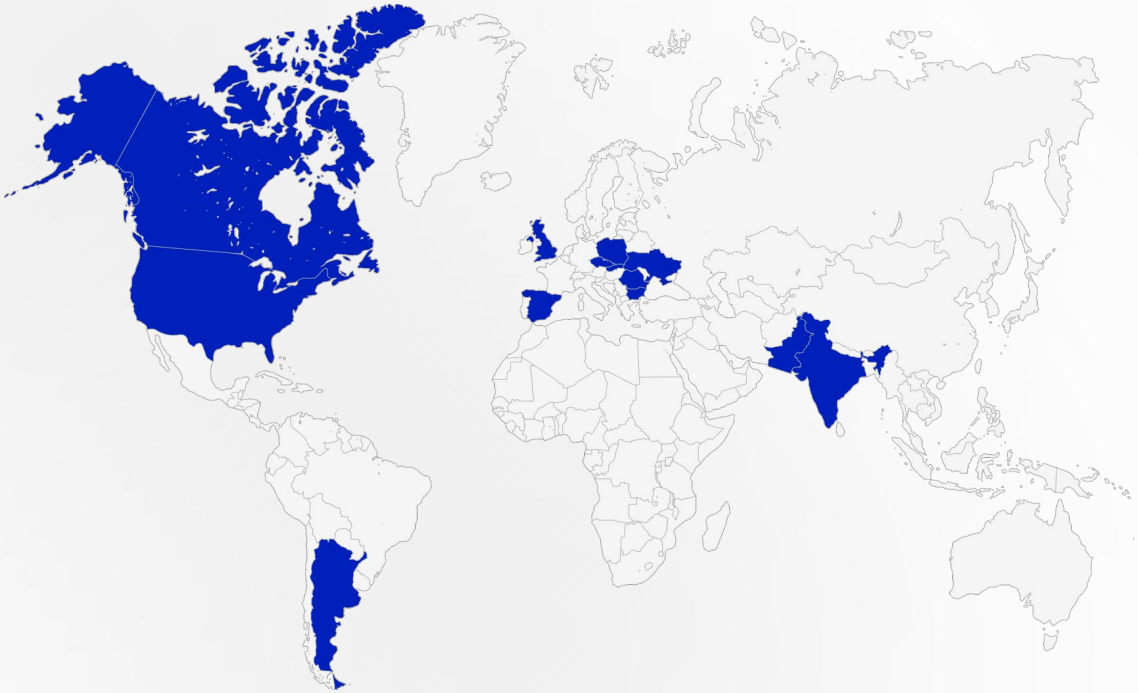
300+
clients

Leading companies choose us:

JUST EAT Takeaway.com · METRO MARKETS · eToro · ZURICH · pay · Greensill · KANTAR · RETAIL · dacadoo

Ciklum's world

CIKLUM



Central & eastern europe

- Bulgaria
- Czech Republic
- Poland
- Romania
- Slovakia
- Spain
- Ukraine
- United Kingdom

Asia

- India
- Pakistan

LATAM

- Argentina
- Uruguay

North America

- Canada
- USA

Meet the speaker

- Senior Embedded Software Engineer @ Ciklum Slovakia
- 12 years of professional experience
- Drone Pilot, 3D Printing Enthusiast, Corgi owner

Adrian Panicek,
Senior Embedded Software Engineerat Ciklum

CIKLUM | SPEAKER'S CORNER

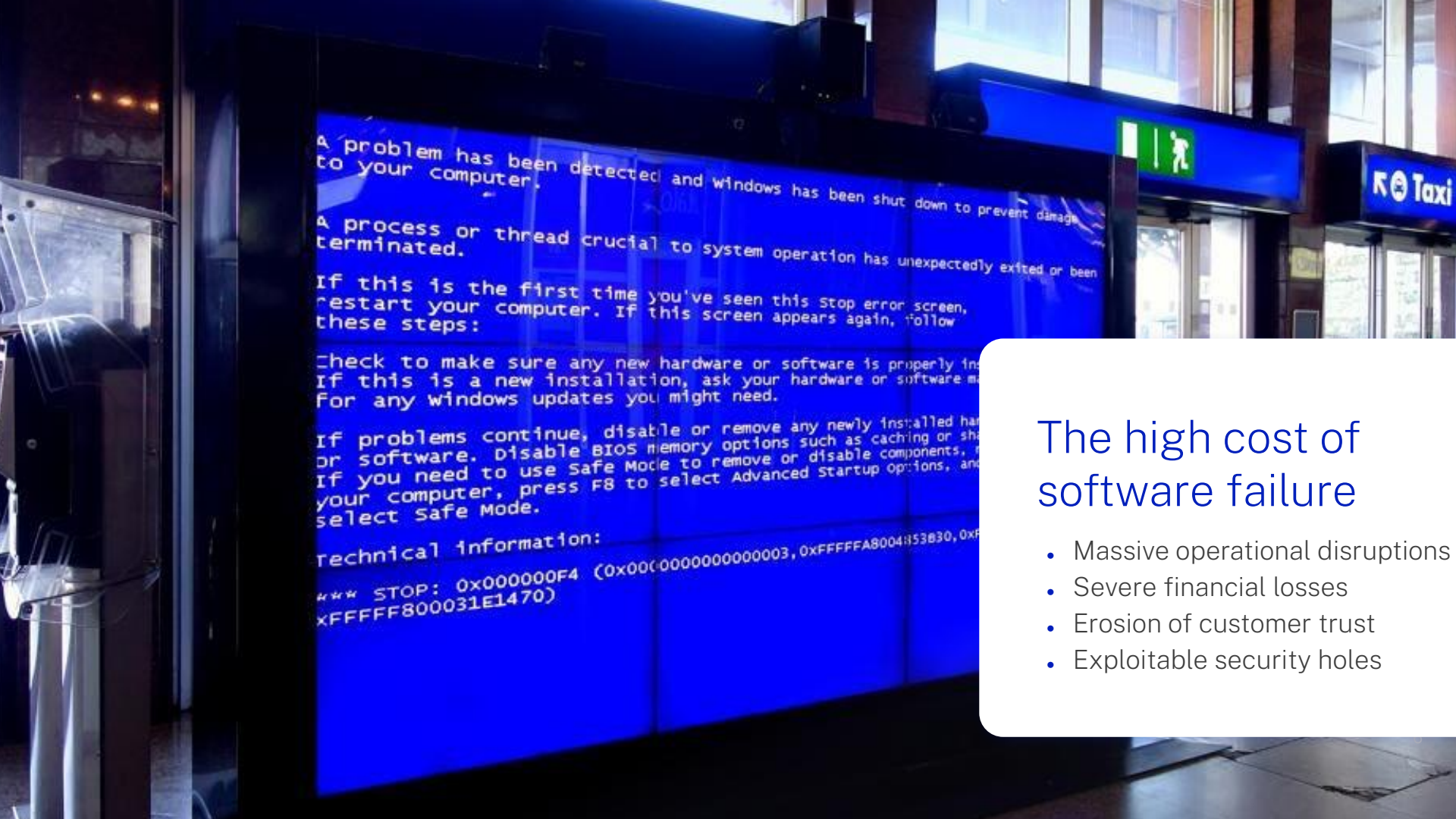
April, 16 | 17:00 (CET) | English

Why can't we afford to ignore Rust?

Software security in numbers

Adrian Panicek

Senior Embedded Software Engineer at Ciklum



The high cost of software failure

- Massive operational disruptions
- Severe financial losses
- Erosion of customer trust
- Exploitable security holes

CIKLUM

Case Study 1: Global Paralysis

(CrowdStrike Outage, July 2024)

- Worldwide System Crashes (BSODs)
- Airlines Grounded
- Hospitals, Banks, Businesses
- Estimated Billions $ in Economic Damage

CIKLUM

Bugs and vulnerabilities cost us money!

- Expensive investigation & fixing cycles
- Costly system downtime
- High incident response & recovery bills
- Brand damage & lost customer trust

What are bugs and vulnerabilities

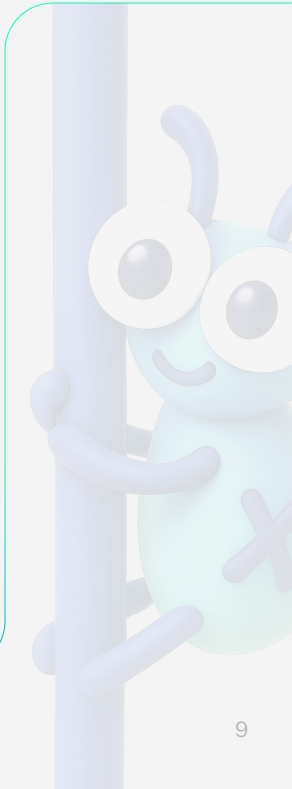
CIKLUM

Bug

- A **flaw in code** causing incorrect or unexpected program behavior
- Might result in **data corruption, outage** or even bodily harm

Vulnerability

- A **weakness in code** enabling attacker to violate security policy
- Might result in **data leaks, identity theft,** financial fraud...



CIKLUM

Case Study 2: Lethal Dose

(Therac-25 Incidents, 1985-1987)

- Massive radiation overdoses delivered
- Caused by software bug
- Cancer patients killed or severely injured
- Landmark failure in software safety & ethics

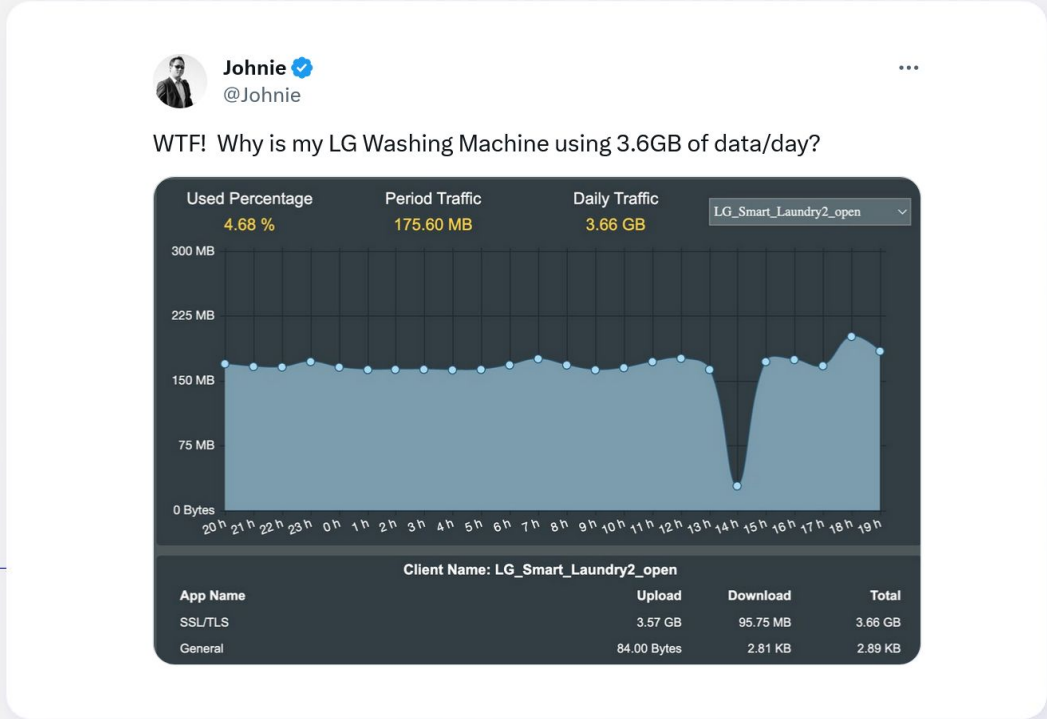
CIKLUM

What caused the issue

- Race condition in the system
- Lack of documentation
- Incoherent error reporting
- Bus factor in development
- Weak tooling

CIKLUM

Don’t let your devices become bots

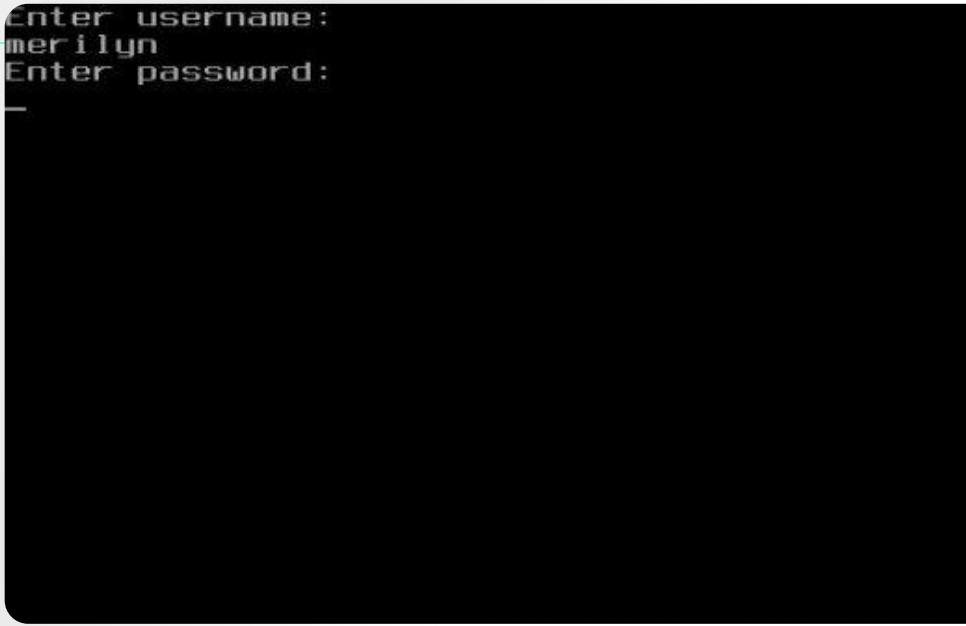


CIKLUM

Case Study 3: Hack With 28 Steps

(CVE-2015-8370, December 2009 - early 2016)

- GRUB bootloader password bypassed
- Triggered by pressing backspace 28 times
- Granted unauthorized GRUB rescue shell access



CIKLUM

What are CVEs

- System to document, evaluate and archive security vulnerabilities
- We collect up to a hundred of vulnerabilities every day
- Each CVE has unique numbering
- cve.org, exploit-db.com

CIKLUM

Just code better

- Common argument states it’s developer’s fault
- The best way for developer to defend is to use better tools
- Not even the best developers are immune to mistakes

Most common manual memory bugs

CIKLUM

- Buffer overflow
- Use-after-free
- Memory leak
- NULL pointer dereference
- Double free
- Heap overflow
- Stack buffer overflow

- Integer overflow/underflow
- Dangling pointer
- Buffer over-read
- Type confusion
- Uninitialized memory Read/Use
- Format string vulnerability
- Out-of-bounds Read/Write

CIKLUM

What is unsafe memory access?

```c
int main(void) {
  const char* source = "Ciklum!";
  char* copy = malloc(7);
  memset(copy, 7, '\0');
  memcpy(copy, source, 7);
  free(copy);

  printf("%s", copy);
  return 0;
}
```

CIKLUM

What is unsafe memory access?

```c
int main(void) {
  const char* source = "Ciklum!";
  char* copy = malloc(7); // Buffer is one byte short!
  memset(copy, 7, '\0'); // Wrong order of parameters!
  memcpy(copy, source, 7); // “source” could be
immutable but C doesn’t support it
  free(copy);

  printf("%s", copy); // Use after free
  return 0;
}
```

Industry is sounding alarms

CIKLUM

Microsoft: 70 percent of all security bugs are memory safety issues

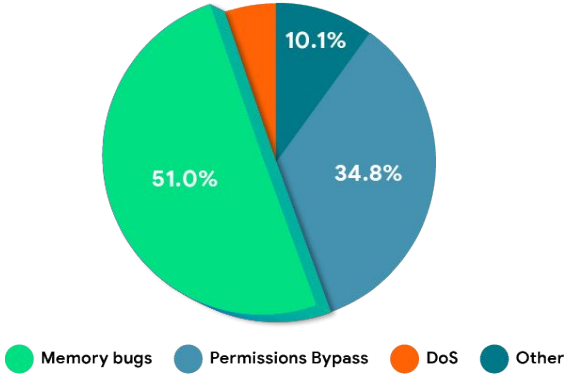
Percentage of memory safety issues has been hovering at 70 percent for the past 12 years.

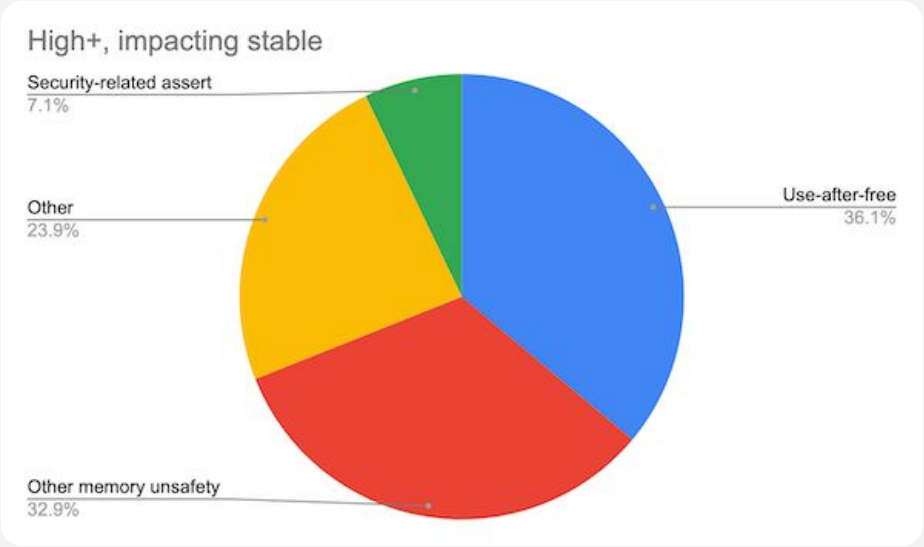
Written by **Catalin Cimpanu**, Contributor
Feb. 11, 2019 at 7:48 a.m. PT

America's Cyber Defense Agency
NATIONAL COORDINATOR FOR CRITICAL INFRASTRUCTURE SECURITY AND RESILIENCE

BLOG

The Urgent Need for Memory Safety in Software Products





We are certainly not immune to memory related bugs, mistakes or vulnerabilities. We count about 40% of our security vulnerabilities to date to have been the direct result of us using C instead of a memory-safe language alternative.

Governments are sounding alarms

CIKLUM

- The Case for Memory Safe Roadmaps, Join efforts Five Eyes (2023)
- How to Protect Against Software Memory Safety Issues, NSA (2022)
- Exploring Memory Safety in Critical Open Source Projects, CISA (2024)

> “Memory management issues have been exploited for decades and are still entirely too common today”
>
> -Neal Ziring, Cybersecurity Technical Director NSA

Are we bashing C?

CIKLUM

- C/C++ is the core of all major operating systems
- C is used for most of high-performance applications
- All garbage collected/higher level languages run on C
- If we want to fix all languages, we need to focus on C first

Linux can’t code better

- Dirty COW (CVE-2016-5195) - Privilege escalation
- Stack clash (CVE-2017-1000364) - Memory corruption
- Baron Samedit (CVE-2021-3156) - LPE in sudo
- BleedingTooth (CVE-2020-12351, CVE-2020-12352, CVE-2020-24490) - Set of vulnerabilities in BLE stack
- ...
- Use-after-free (CVE-2017-7308 and many more!)
- Null pointer dereference (Numerous and ongoing!)

Even The Best Make Mistakes

CIKLUM

CIKLUM

Even the best make mistakes

“This is an ancient bug that was actually attempted to be fixed once (badly) by me 11 years ago”

Linus Torvalds on DirtyCOW

CIKLUM

The need for better tooling

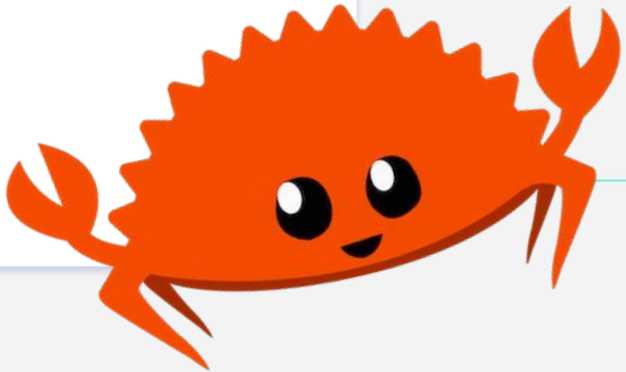
- Many of the bugs and vulnerabilities are preventable
- Best practices
- Defensive programming
- Static analysis
- Wherever IO, multithreading or magic happens, bugs are inevitable

Here comes Rust!

CIKLUM

- Broad-level
- Cross-Platform
- Community driven
- Open source
- Built on LLVM backend

- Memory safe
- C-Level performance
- Thread safe
- Reliable



Rust timeline

CIKLUM

Rust won multiple Stack Overflow Developer Surveys for the most favorite language, maintaining position for several years

2006

Rust started as a personal project by **Graydon Hoare**

2009

Mozilla began sponsoring Rust development

2019

Rust 1.0 was released, marking the language as stable

2020

Dropbox, Cloudflare, and Microsoft partially adopted Rust

2021

Rust was adopted for kernel development in **Linux**

Industry trusts Rust!

CIKLUM

How does Rust do it?

CIKLUM

- Ownership model
- Immutability by default
- Option monad instead of null pointers
- Thread safe operations
- Smart pointers
- Integrated unit testing
- Integrated package management
- Strict type system

Ownership model

CIKLUM

```
void main() {
    const char* s1 = "hello";
    const char* s2 = s1;

    printf("%s world!", s1);
}
```

```
fn main() {
    let s1 = String::from("hello");
    let s2 = s1;

    println!("{s1}, Ciklum!");
}
```

Ownership model

CIKLUM

```
error[E0382]: borrow of moved value: `s1`
 --> src/main.rs:5:15
  |
2 |     let s1 = String::from("hello");
  |         -- move occurs because `s1` has type `String`, which does not implement the
`Copy` trait
3 |     let s2 = s1;
  |              -- value moved here
4 |
5 |     println!("{s1}, world!");
  |              ^^^^ value borrowed here after move
```

Immutability

CIKLUM

```c
void main() {
    int x = 5;
    printf("The value of x is: %d", x);
    x = 6;
    printf("The value of x is: %d", x);
}
```

```rust
fn main() {
    let x = 5;
    println!("The value of x is: {x}");
    x = 6;
    println!("The value of x is: {x}");
}
```

Variables **are immutable by default**

CIKLUM

```
error[E0384]: cannot assign twice to immutable variable `x`
 --> src/main.rs:4:5
  |
2 |   let x = 5;
  |       - first assignment to `x`
3 |   println!("The value of x is: {x}");
4 |   x = 6;
  |   ^^^^^ cannot assign twice to immutable variable
  |
help: consider making this binding mutable
  |
2 |   let mut x = 5;
  |       +++
```

No NULL pointers

CIKLUM

```
void main() {
    const char* s1 = "hello";
    const char* s2 = s1;

    printf("%s world!", s1);
}
```

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

“I call it my billion-dollar mistake...”

Tony Hoare, creator of null pointer

Safety meets savings

CIKLUM

- **Eliminates bug categories:** Compile-time checks eradicate memory safety errors and data races.
- **Reduced downtime:** Fewer runtime crashes and unexpected behaviors lead to more reliable systems.
- **Lower security risks:** Prevents many common CVEs exploited via memory unsafety, reducing incident response and patching costs.
- **Enhanced productivity:** Clear compiler messages, integrated tooling, and less time spent debugging memory issues boost developer efficiency.

Rust is safe **by default**

CIKLUM

- ~~Buffer Overflow~~
- ~~Use-After-Free~~
- ~~Memory Leak~~
- ~~NULL Pointer Dereference~~
- ~~Double Free~~
- Heap Overflow
- Stack Buffer Overflow

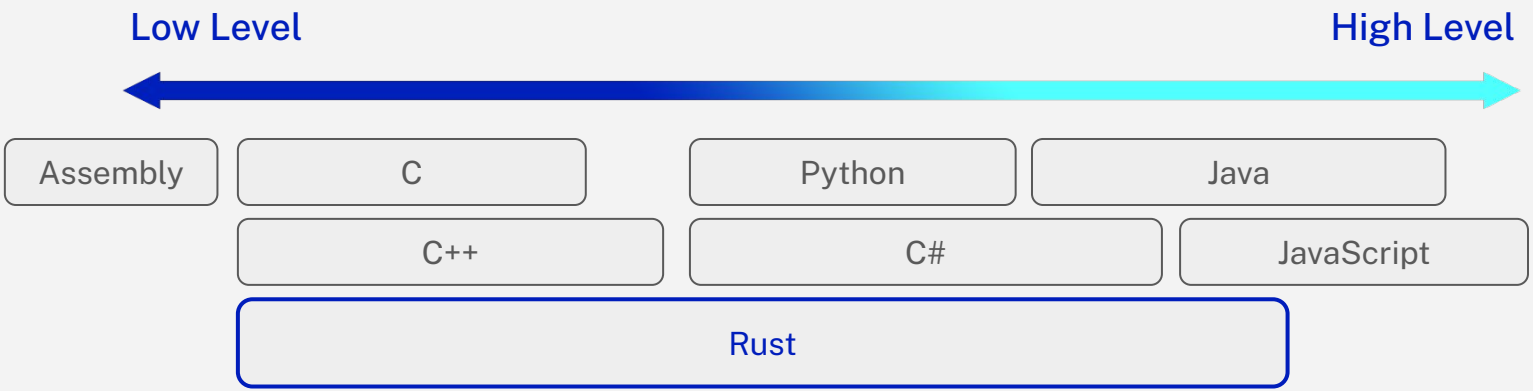
- Integer Overflow/Underflow
- ~~Dangling Pointer~~
- ~~Buffer Over-read~~
- ~~Type Confusion~~
- ~~Uninitialized Memory Read/Use~~
- ~~Format String Vulnerability~~
- ~~Out-of-Bounds Read/Write~~

Performance without compromise

CIKLUM

- **Native speed:** Compiles directly to efficient machine code, rivaling C/C++
- **Zero-cost abstractions:** High-level language features often compile down with no runtime overhead
- **No garbage collector:** Predictable performance without GC pauses; fine-grained control over memory
- **Fearless concurrency:** Build fast, parallel applications without the typical data race nightmares

Broad-level?



- Direct memory manipulation
- Low abstraction
- High performance
- Complicated development
- Compiled

- Advanced data structures
- High abstraction
- Poor performance
- Easy development
- Interpreted

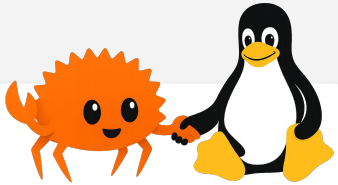
Integrating Rust incrementally

CIKLUM

- **Plays well with others:** Excellent Foreign Function Interface (FFI) allows calling C/C++ code from Rust and vice-versa.
- **Targeted rewrites:** Start by rewriting performance-critical or security-sensitive modules in existing C/C++ projects.
- **New developments:** Ideal for new tools, microservices, embedded systems, and backend services.
- **Gradual adoption:** Teams can learn and integrate Rust at their own pace.

Rust integration in Linux

CIKLUM



- **Official support:** Since Kernel 6.1 (Late 2022)
- **Infrastructure maturing:** Core framework, builds, basic abstractions
- **First real use cases merged/In progress:** The Asahi GPU, WiFi drivers and many small components
- **Cautious but steady progress:** Development is active, backed by sponsors (like Google). Rust code is still very small relative to C, but it's growing meaningfully.

Considerations on the integration

CIKLUM

- **Learning curve:** The ownership and borrow checker concepts require an initial investment to understand well.
- **Talent acquisition:** While growing fast, the Rust talent pool is still maturing compared to C/C++.
- **Compilation times:** Can sometimes be longer, though significant improvements are ongoing.
- **Ecosystem maturity:** While broad, specific niche domains might have fewer established libraries than legacy ecosystems.

Rust and **developers**

CIKLUM

- **Most loved/Admired language (2016-2024):** As per Stack Overflow Developer Surveys
- **Great documentation:** Documentation serves as primary learning point for Rust developers as per 2024 State Of Rust Survey
- **Better compensation:** Rust developers make ~17% more then C developers as per 2024 Stack Overflow Developer Survey

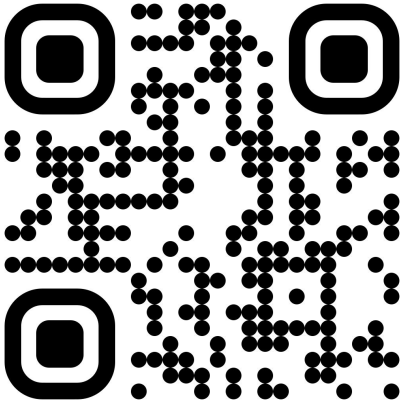
CIKLUM

Embrace the safer future

- **Explore:** Dive into the official Rust Book (rust-lang.org).
- **Experiment:** Initiate a small pilot project – a command-line tool, a small web service, or rewrite a troublesome module.
- **Engage:** Join the Rust community forums, Discord, or local meetups.
- **Educate:** Invest in team training and knowledge sharing.
- **Evaluate:** Seriously consider Rust for new projects where safety, performance, and concurrency are paramount.

CIKLUM

Demo



https://cveroulette.com/

Thank you!

CIKLUM

Any questions?

CIKLUM

Share your feedback!

CIKLUM

Join our team

Thank you!