CASA
CYBER SECURITY IN THE AGE OF LARGE-SCALE ADVERSARIES

“I wouldn’t want my unsafe code to run my pacemaker”

An Interview Study on the Use, Comprehension, and Perceived Risks of Unsafe Rust

Sandra Höltervennhoff[L], Philip Klostermeyer[C], Noah Wöhler[C], Yasemin Acar[P], and Sascha Fahl[C]

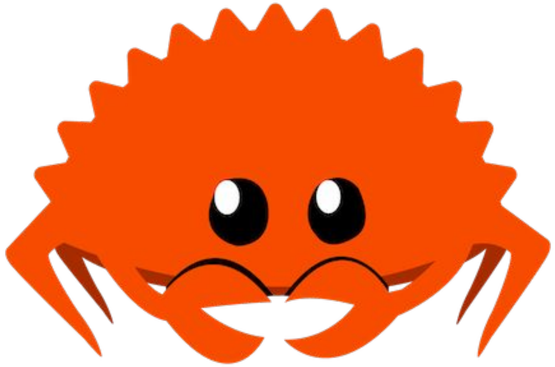
[L] Leibniz University Hannover

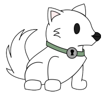
[C] CISPA Helmholtz Center for Information Security

[P] Paderborn University & The George Washington University

Rust is memory safe but still efficient

- Most low-level programming languages impose the burden of secure programming on the developer
- Rust provides **concurrency and memory safety** while being fast and performant
- Rust offers security guarantees not bypassable in "normal" Rust

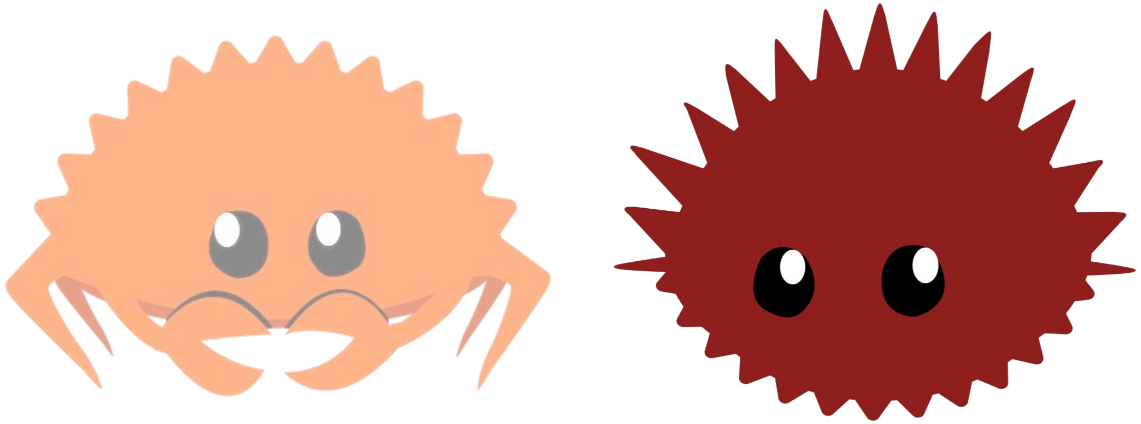




Specific actions can only be used in unsafe Rust

- Five unsafe actions

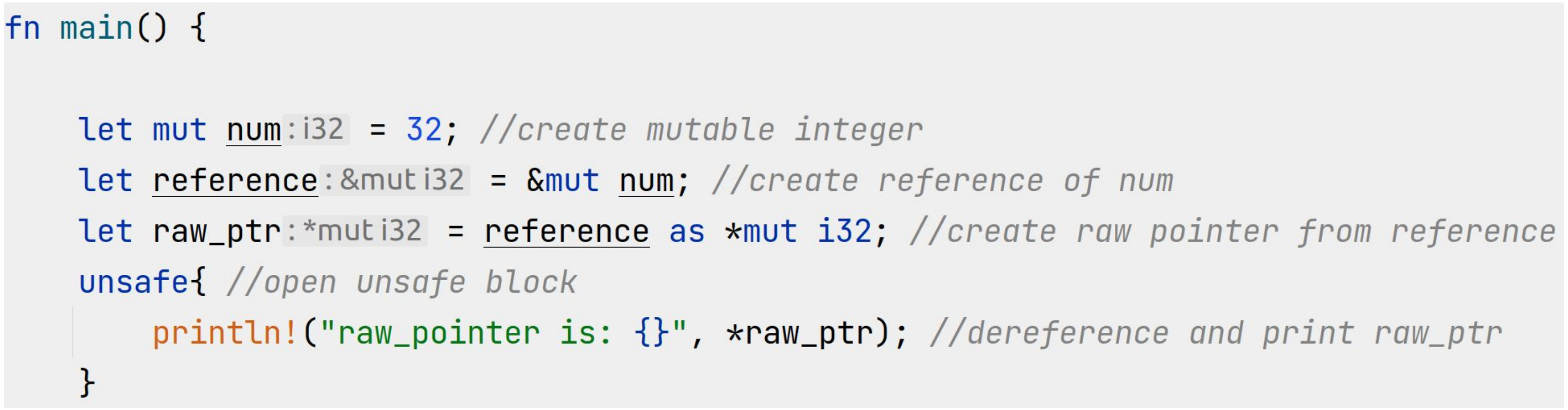
  (e.g., assessing raw memory, calling unsafe functions)



Unsafe actions cannot be checked

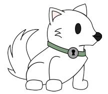
- These unsafe actions cannot be checked by the compiler

→ Memory safety guarantees are not enforced



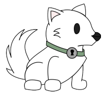
```rust
fn main() {

    let mut num: i32 = 32; //create mutable integer
    let reference: &mut i32 = &mut num; //create reference of num
    let raw_ptr: *mut i32 = reference as *mut i32; //create raw pointer from reference
    unsafe{ //open unsafe block
        println!("raw_pointer is: {}", *raw_ptr); //dereference and print raw_ptr
    }
```



Little research on developers’ perspective on unsafe Rust

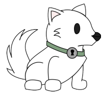
- Research analyzing the “as-is” state and isolation of unsafe code
- Only little research on how developers approach writing, reviewing, and testing unsafe code fragments



Little research on developers' perspective on unsafe Rust

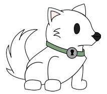
- Research analyzing the "as-is" state and isolation of unsafe code
- Only little research on how developers approach writing, reviewing, and testing unsafe code fragments

→ **Interview study with Rust developers**



Our research questions

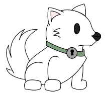
RQ1 What are common practices of **deciding for and implementing** software using **unsafe Rust**?



Our research questions

RQ1 What are common practices of deciding for and implementing software using unsafe Rust?

RQ2 How do developers assess **unsafe Rust's features, limitations,** and **security risks**?

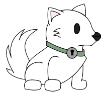


Our research questions

RQ1 What are common practices of deciding for and implementing software using unsafe Rust?

RQ2 How do developers assess unsafe Rust's features, limitations, and security risks?

RQ3 What security **code reviewing** and **testing** practices **around unsafe Rust** are used by developers?



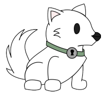
Our research questions

RQ1 What are common practices of deciding for and implementing software using unsafe Rust?

RQ2 How do developers assess unsafe Rust's features, limitations, and security risks?

RQ3 What security code reviewing and testing practices around unsafe Rust are used by developers?

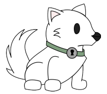
RQ4 How do developers experience **security incidents** as the result of incorrect use **of unsafe Rust**?



Interviews with Rust developers

- **Online** interviews
- Duration approx. 1 h
- 7 thematic blocks
- **26 Rust developers**

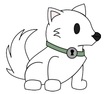
1. Background and Demographics
2. Comprehension and Experience for Unsafe Code
3. Testing and Tooling
4. Security Policies and Guidelines for Unsafe Code
5. Code Reviewing Unsafe Code
6. Incidents and Threat Model Using Unsafe Code
7. Conclusion and Outro



Many try to avoid unsafe code

- Most **only** used unsafe code **if really necessary**
- Some participants resorted to unsafe code if performance was (noticeably) better
- Participants reported a **thoughtful writing process** (e.g., safe interfaces, safety comments)

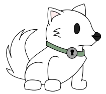
"I wouldn't want my unsafe code to run my pacemaker." - P07



Crates that encapsulate unsafe code are popular

- Half of our participants **used crates** so that they do not have to write unsafe code themselves
- Participants often picked unsafe crates by their **reputation or at most meta-verified** them

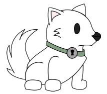
"What we do mostly, we just look at when was the last update, how many downloads does it have? So if it's popular and maintained, and it actually adds value, that's the only extent we don't verify them any further." - P14



Some use unsafe code without diligent care

- A few participants **knew** that their unsafe code lead to undefined behavior
- Two reasons were too many lines of unsafe code or projects perceived as less important/security-critical

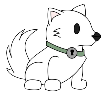
"Once you start running unsafe everywhere, it just starts to become annoying. So then we just say, 'Oh yes, this unsafe block count on the caller doing it right.'" - P05



Unsafe code seen as a beacon

- Half of our participants perceived unsafe Rust as code blocks, where developers can **pay close attention** to security
- Many participants mentioned that unsafe Rust could be **scary** and error-prone
- Some participants **incorrectly** thought that compiler checks were turned off

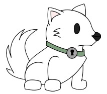
"[T]here's still a lot of things that nobody really knows if what you're doing is sound. Sometimes, theoretically, it's not sound, but in practice, nobody has ever encountered an issue. So we do it anyway." - P25



Policies on how to use unsafe code are rare

- Developers trusted their **common sense**
- Some participants named constraints (e.g., code reviews, team discussion, lead maintainer deciding)

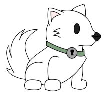
> "Typically, in a project when the topic of unsafe code comes up, it is because it is clear that something should be solved with unsafe code." - P04



Participants often try to test unsafe more rigorously

- Testing unsafe was sometimes reported as **complex or not possible**
- A few participants found security-related tools to be **cumbersome**

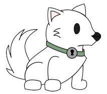
"One of the great things about Rust is that a lot of the tooling just works. [. . .] There's a high level of laziness when it comes to integrating new tooling. I don't really like to go jump through a ton of hoops to get something set up." - P25



No severe security incidents

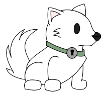
- Most reported bugs in unsafe code were minor
- Participants **did not** necessarily **connect** undefined behavior or bugs with security vulnerabilities

"My understanding is that not all unsoundness is necessarily a vulnerability, right? There are unsoundnesses that are not large threats, depending on, of course on the threat model." -P08



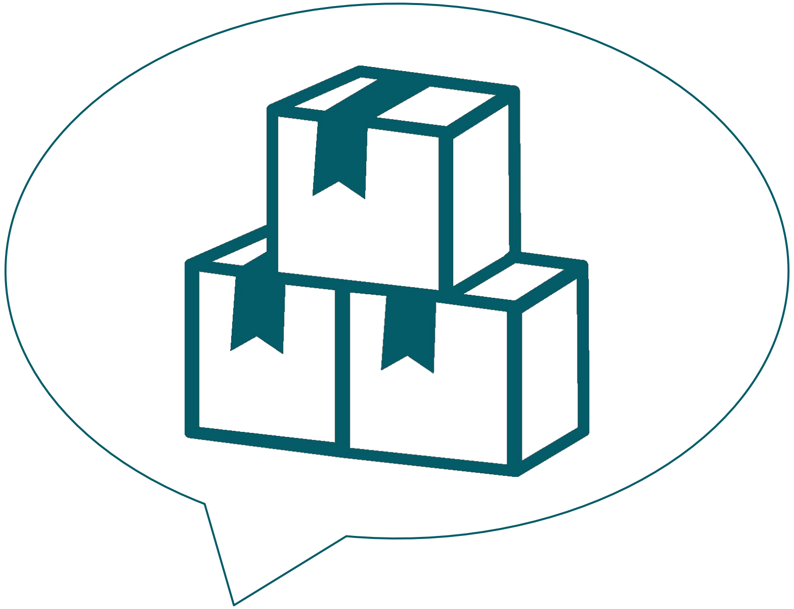
Unsafe code is mostly used securely… but not always

- The Rust community successfully communicates risks around unsafe code
- Nevertheless, participants became inattentive or annoyed when unsafe code was frequently used
- Experience might not always be sufficient to handle unsafe code



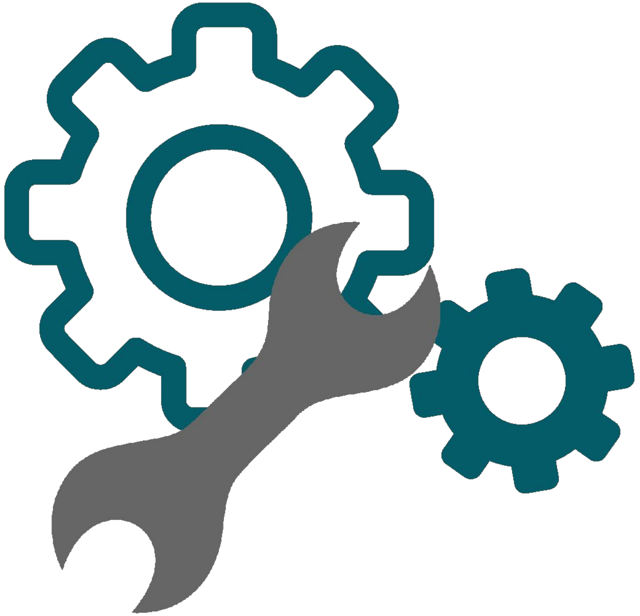
Communication should be further improved

- A big factor for the future is the **packaging of information**



Deploying simplified onboarding assistants

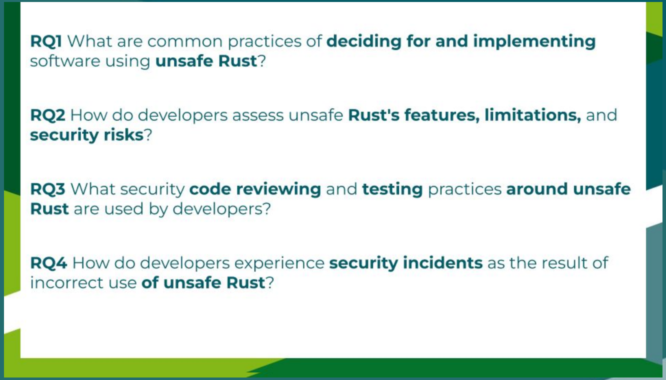
- E.g., configuration wizards



Assist developers in choosing the right crate

- **Integrating** such metrics into crates.io

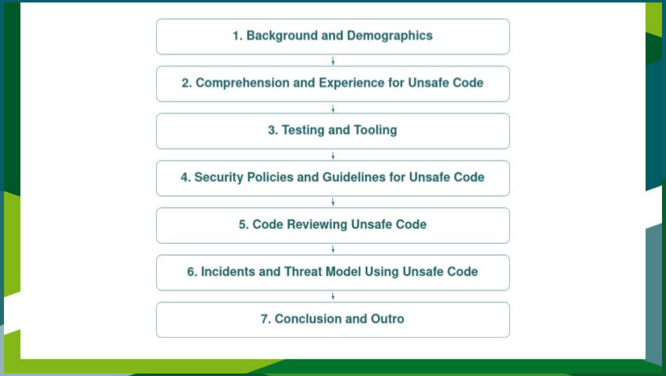
Thank you for your attention!

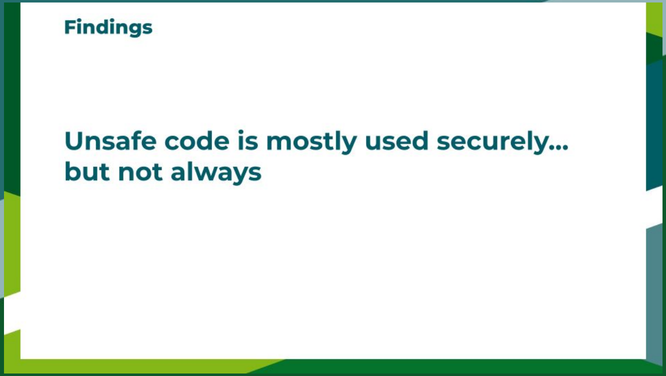

RQ1 What are common practices of **deciding for and implementing** software using **unsafe Rust**?

RQ2 How do developers assess unsafe **Rust's features, limitations,** and **security risks**?

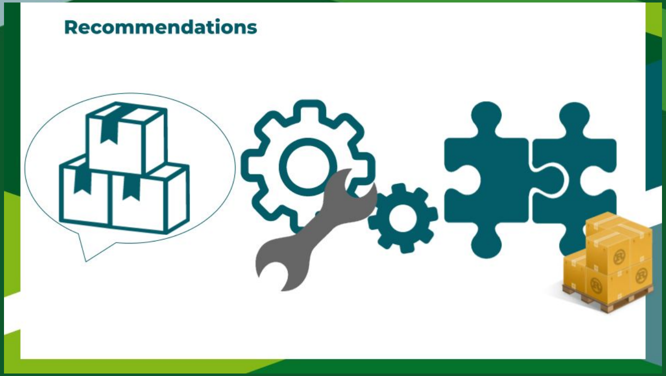
RQ3 What security **code reviewing** and **testing** practices **around unsafe Rust** are used by developers?

RQ4 How do developers experience **security incidents** as the result of incorrect use **of unsafe Rust**?


1. Background and Demographics
2. Comprehension and Experience for Unsafe Code
3. Testing and Tooling
4. Security Policies and Guidelines for Unsafe Code
5. Code Reviewing Unsafe Code
6. Incidents and Threat Model Using Unsafe Code
7. Conclusion and Outro


Findings

Unsafe code is mostly used securely… but not always


Recommendations

Sandra Höltervennhoff
Contact: hoeltervennhoff@sec.uni-hannover.de