Rudra: Finding Memory Safety Bugs in Rust at the Ecosystem Scale

Yechan Bae, Youngsuk Kim, Ammar Askar, Jungwon Lim, Taesoo Kim

Georgia Tech Systems Software & Security Lab (SSLab)

SOSP 2021

Rust for safe systems programming

Rust for safe systems programming

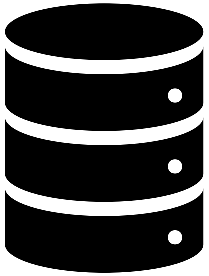
Google

facebook

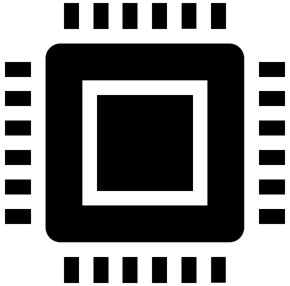
amazon

Discord

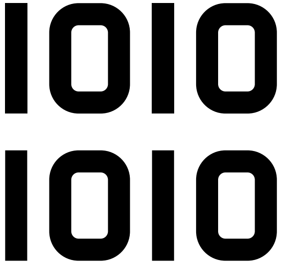
Dilemma: Safety vs Control



Memory-mapped I/O



Hardware Abstraction



OS Interaction

Dilemma: Safety vs Control

Memory-mapped I/O

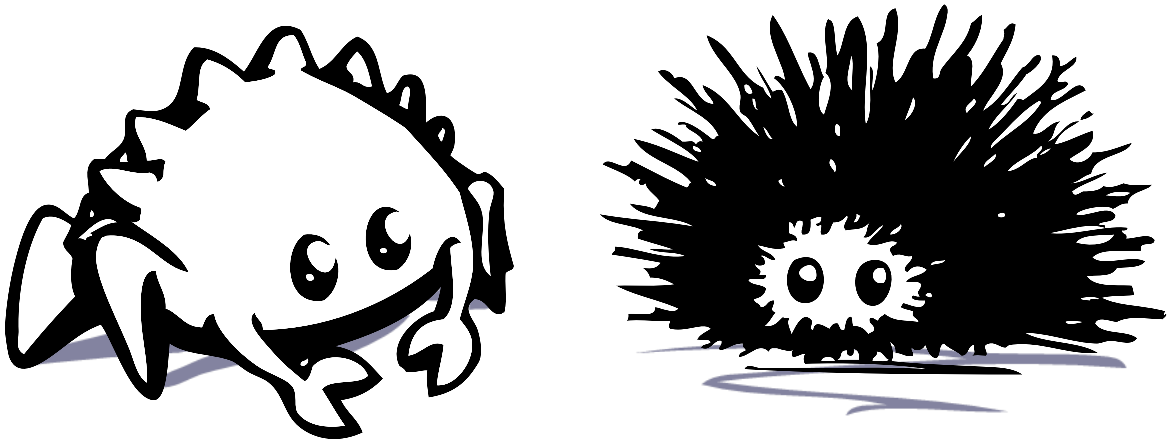
Hardware Abstraction

OS Interaction

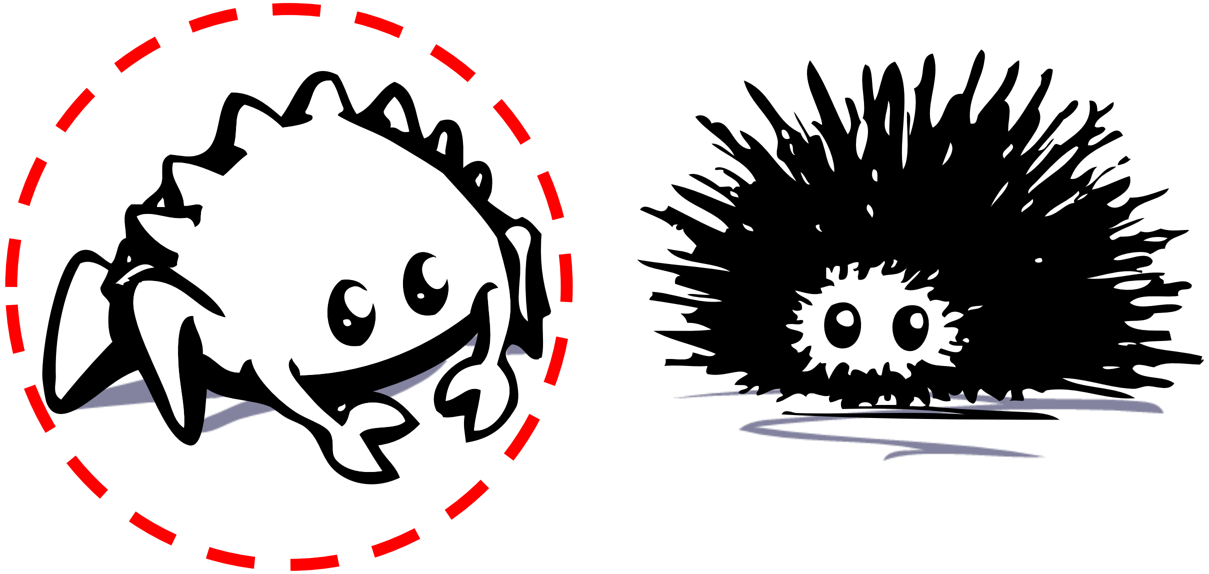
Escape hatch: unsafe Rust

Safe and Unsafe Rust

Safe and Unsafe Rust

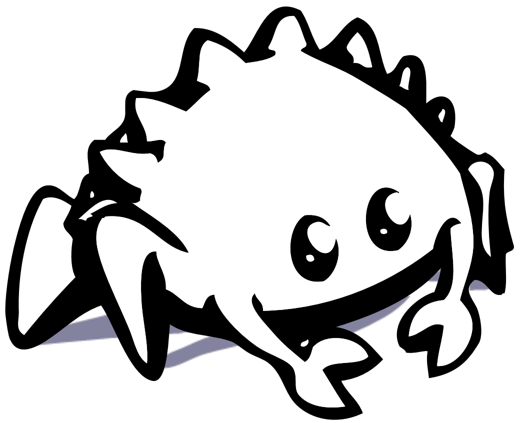


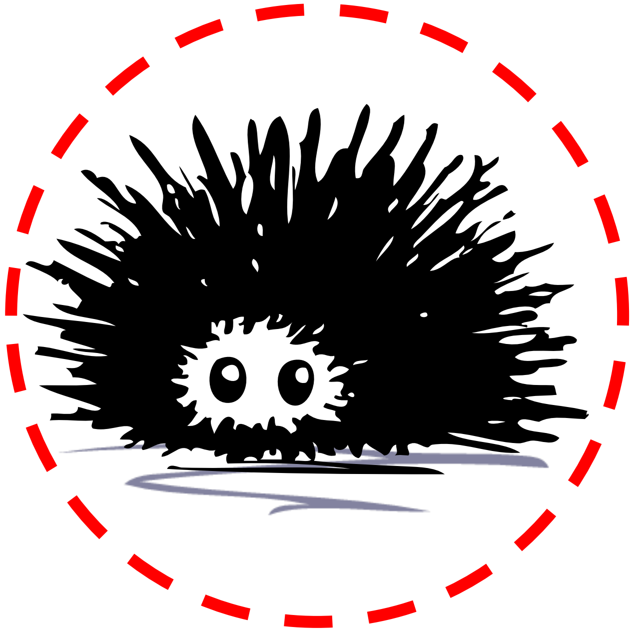
Safe and Unsafe Rust



If a program is written entirely in safe Rust,
the Rust compiler automatically guarantees the memory safety

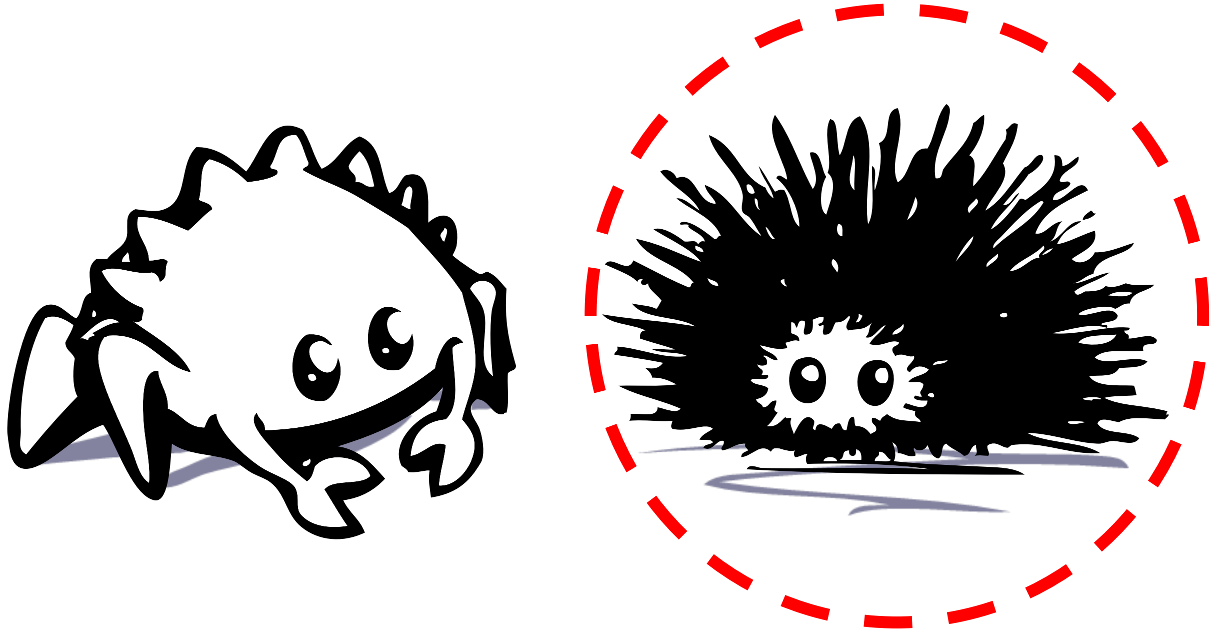
Safe and Unsafe Rust





If a program contains unsafe Rust,
the programmer needs to guarantee the memory safety

Safe and Unsafe Rust



Memory safety of a Rust program depends on the correctness of all unsafe code it contains

Two Ways of Using Unsafe Rust

```
unsafe fn access_unchecked(index: usize) {
    ...
}
```

1. Unsafe API can be directly exposed to users

- Caller is responsible for providing a correct argument (e.g., in-bound index)

```
fn access(index: usize) {
    assert!(index < self.len());
    unsafe { access_unchecked(index); }
}
```

2. Unsafe API can be encapsulated in safe API

- API designer guarantees that <u>this API never causes memory safety bugs</u>

Two Ways of Using Unsafe Rust

```
unsafe fn access_unchecked(index: usize) {
    ...
}
```

1. Unsafe API can be directly exposed to users

- Caller is responsible for providing a correct argument (e.g., in-bound index)

Our target

```
fn access(index: usize) {
    assert!(index < self.len());
    unsafe { access_unchecked(index); }
}
```

2. Unsafe API can be encapsulated in safe API

- API designer guarantees that <u>this API never causes memory safety bugs</u>

Rudra: A Static Analyzer for Unsafe Rust

- We identified three common bug patterns in unsafe Rust
- We devised two new algorithms to detect them
- We implemented a static analyzer named Rudra, that can scale to the entire Rust ecosystem (43k packages / 6.5 hours)
- Found more than half of the memory safety bugs known to the Rust security advisory database (RustSec)
    - 76 CVEs and 112 RustSec advisories
    - Including two memory safety bugs in the Rust standard library

The Three Bug Patterns

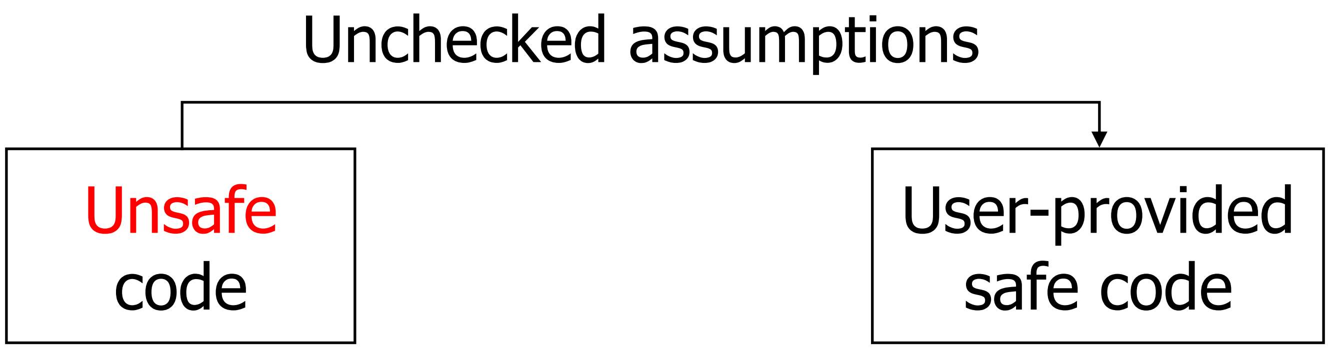
1. **Panic safety bug**
   - Incorrect resource deallocation in compiler-inserted invisible code paths
2. **Higher-order invariant bug**
   - Unchecked assumptions on user-provided higher-order values
3. **Send/Sync variance bug**
   - Incorrect condition for manual thread safety assertions

The Three Bug Patterns

1. **Panic safety bug**
   - Incorrect resource deallocation in compiler-inserted invisible code paths
2. **Higher-order invariant bug**
   - Unchecked assumptions on user-provided higher-order values
3. **Send/Sync variance bug**
   - Incorrect condition for manual thread safety assertions

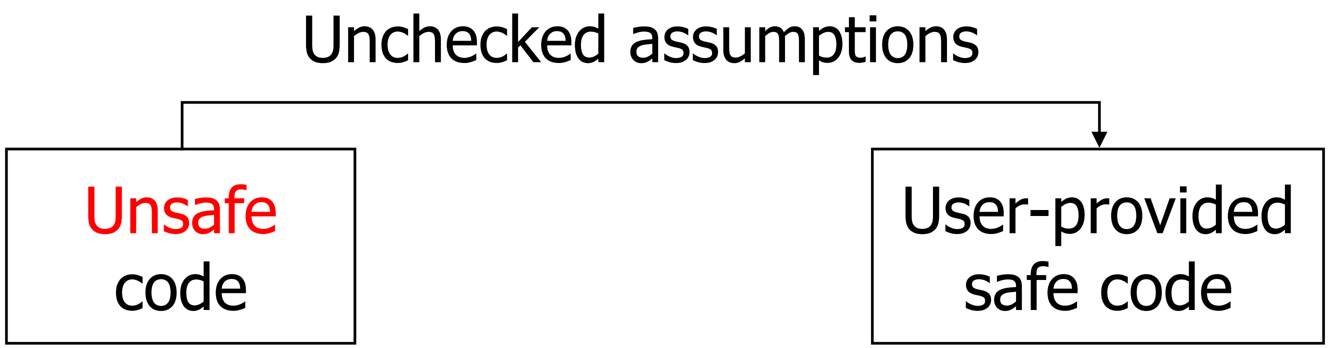
Higher-Order Invariant Bug

Unchecked assumptions on user-provided higher-order values



Higher-Order Invariant Bug

Unchecked assumptions on user-provided higher-order values

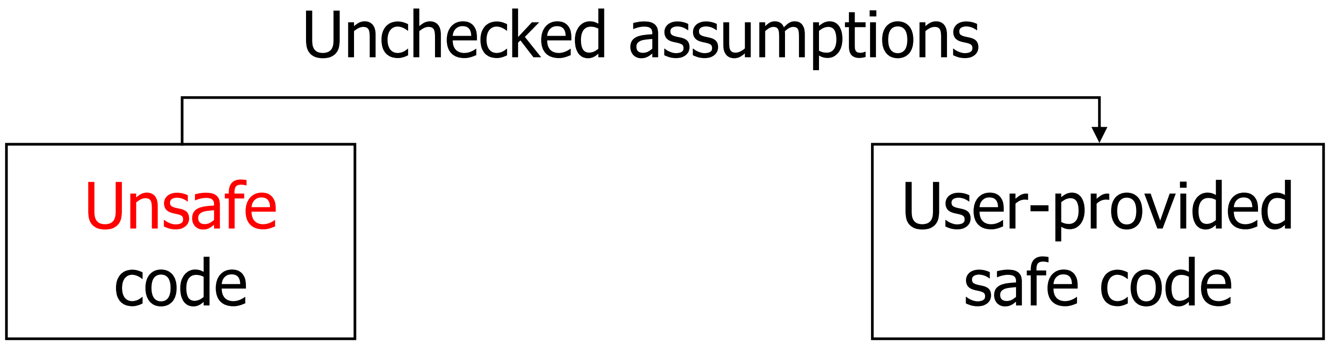


Invariant

```
fn access(index: usize)
```

Higher-Order Invariant Bug

Unchecked assumptions on user-provided higher-order values

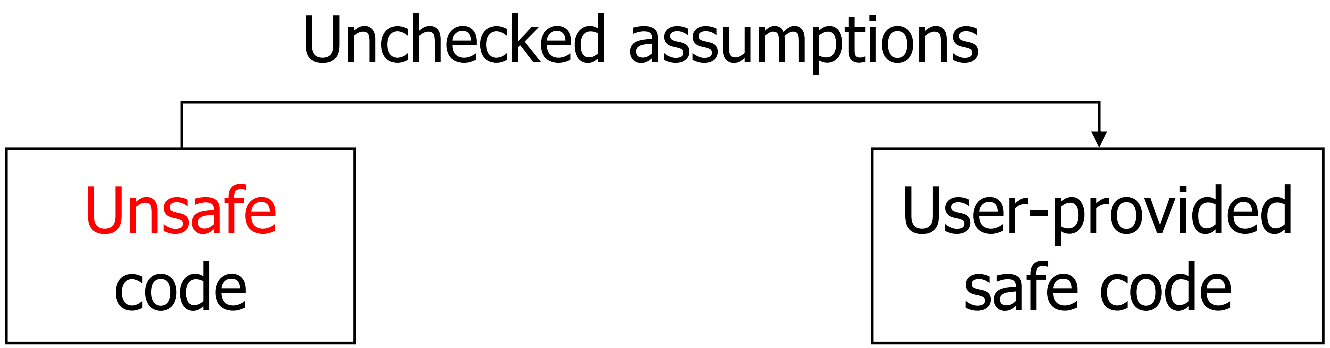


Invariant

```
fn access(index: usize)
```

Higher-Order Invariant Bug

Unchecked assumptions on user-provided higher-order values



Invariant

```
fn access(index: usize)
```

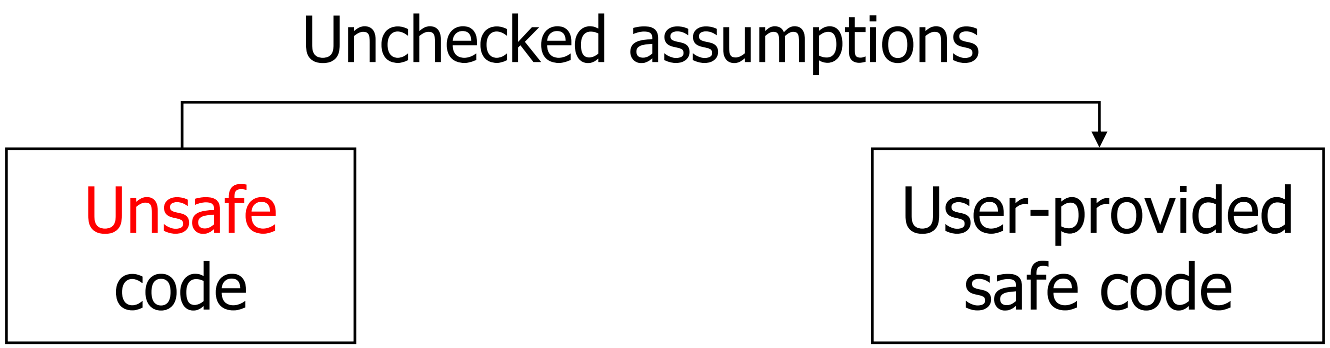
Higher-Order Invariant

```
fn sort_by<F>(&mut self, compare: F)
```

F: comparator function (closure)

Higher-Order Invariant Bug

Unchecked assumptions on user-provided higher-order values



Higher-Order Invariant

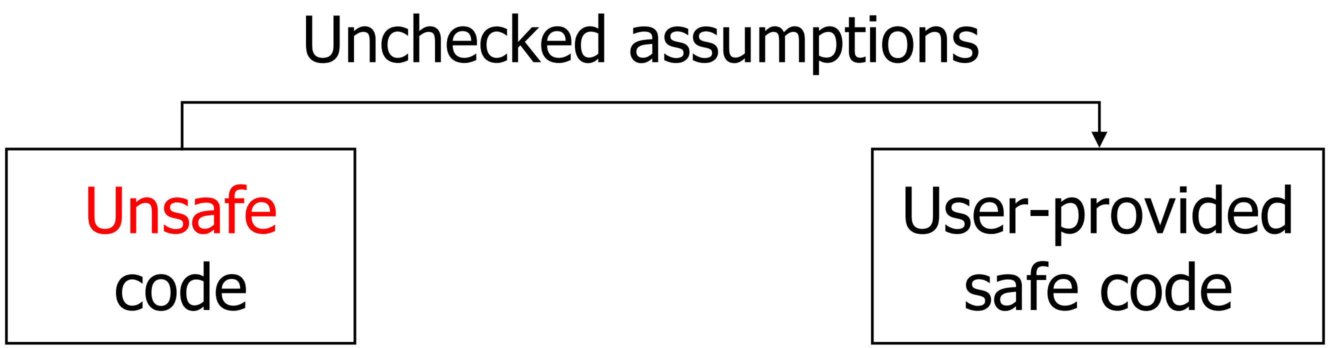
```
fn sort_by<F>(&mut self, compare: F)
```

F: comparator function (closure)

(1) Time-of-check to time-of-use
(2) Logical assumptions (e.g., transitivity)

Higher-Order Invariant Bug

Unchecked assumptions on user-provided higher-order values



Higher-Order
Invariant

```
fn sort_by<F>(&mut self, compare: F)
```

F: comparator function (closure)

(1) Time-of-check to time-of-use
(2) Logical assumptions (e.g., transitivity)

Example: CVE-2020-36323, a higher-order invariant bug in string join()
Found by Rudra in the Rust standard library

Bug Example

```
// join(&["a", "b", "c"], "|") => "a|b|c"
fn join<T>(array: &[T], sep: &str) -> String
    where T: Convert<str>
{
    // code that handles array.len() == 0 or 1
    ...

    let len = sep.len() * (array.len() - 1)
        + array.iter().map(|s| s.convert().len()).sum();

    let mut result = String::with_capacity(len);

    unsafe {
        // contains uninitialized bytes
        let mut buf = result.get_unchecked_mut(..len);

        buf.copy_and_advance(array[0].convert());
        for s in &array[1..] {
            buf.copy_and_advance(sep);
            buf.copy_and_advance(s.convert());
        }

        result.set_len(len);
        return result;
    }
}
```

CVE-2020-36323

A higher-order invariant bug in string join()

Found by Rudra in the Rust standard library

(Code simplified and renamed for presentation)

Bug Example

```
// join(&["a", "b", "c"], "|") => "a|b|c"
fn join<T>(array: &[T], sep: &str) -> String
    where T: Convert<str>
{
    // code that handles array.len() == 0 or 1
    ...

    let len = sep.len() * (array.len() - 1)
        + array.iter().map(|s| s.convert().len()).sum();

    let mut result = String::with_capacity(len);

    unsafe {
        // contains uninitialized bytes
        let mut buf = result.get_unchecked_mut(..len);

        buf.copy_and_advance(array[0].convert());
        for s in &array[1..] {
            buf.copy_and_advance(sep);
            buf.copy_and_advance(s.convert());
        }

        result.set_len(len);
        return result;
    }
}
```

CVE-2020-36323

A higher-order invariant bug in string join()

Found by Rudra in the Rust standard library

(Code simplified and renamed for presentation)

Bug Example

```
// join(&["a", "b", "c"], "|") => "a|b|c"
fn join<T>(array: &[T], sep: &str) -> String
    where T: Convert<str>
{
    // code that handles array.len() == 0 or 1
    ...

    let len = sep.len() * (array.len() - 1)
        + array.iter().map(|s| s.convert().len()).sum();

    let mut result = String::with_capacity(len);

    unsafe {
        // contains uninitialized bytes
        let mut buf = result.get_unchecked_mut(..len);

        buf.copy_and_advance(array[0].convert());
        for s in &array[1..] {
            buf.copy_and_advance(sep);
            buf.copy_and_advance(s.convert());
        }

        result.set_len(len);
        return result;
    }
}
```

CVE-2020-36323

A higher-order invariant bug in string join()

Found by Rudra in the Rust standard library

(Code simplified and renamed for presentation)

Bug Example

```
// join(&["a", "b", "c"], "|") => "a|b|c"
fn join<T>(array: &[T], sep: &str) -> String
    where T: Convert<str>
{
    // code that handles array.len() == 0 or 1
    ...

    let len = sep.len() * (array.len() - 1)
        + array.iter().map(|s| s.convert().len()).sum();

    let mut result = String::with_capacity(len);

    unsafe {
        // contains uninitialized bytes
        let mut buf = result.get_unchecked_mut(..len);

        buf.copy_and_advance(array[0].convert());
        for s in &array[1..] {
            buf.copy_and_advance(sep);
            buf.copy_and_advance(s.convert());
        }

        result.set_len(len);
        return result;
    }
}
```

CVE-2020-36323

A higher-order invariant bug in string join()

Found by Rudra in the Rust standard library

(Code simplified and renamed for presentation)

Bug Example

```
// join(&["a", "b", "c"], "|") => "a|b|c"
fn join<T>(array: &[T], sep: &str) -> String
    where T: Convert<str>
{
    // code that handles array.len() == 0 or 1
    ...

    let len = sep.len() * (array.len() - 1)
        + array.iter().map(|s| s.convert().len()).sum();

    let mut result = String::with_capacity(len);

    unsafe {
        // contains uninitialized bytes
        let mut buf = result.get_unchecked_mut(..len);

        buf.copy_and_advance(array[0].convert());
        for s in &array[1..] {
            buf.copy_and_advance(sep);
            buf.copy_and_advance(s.convert());
        }

        result.set_len(len);
        return result;
    }
}
```

CVE-2020-36323

A higher-order invariant bug in string join()

Found by Rudra in the Rust standard library

(Code simplified and renamed for presentation)

Bug Example

```rust
// join(&["a", "b", "c"], "|") => "a|b|c"
fn join<T>(array: &[T], sep: &str) -> String
    where T: Convert<str>
{
    // code that handles array.len() == 0 or 1
    ...

    let len = sep.len() * (array.len() - 1)
        + array.iter().map(|s| s.convert().len()).sum();

    let mut result = String::with_capacity(len);

    unsafe {
        // contains uninitialized bytes
        let mut buf = result.get_unchecked_mut(..len);

        buf.copy_and_advance(array[0].convert());
        for s in &array[1..] {
            buf.copy_and_advance(sep);
            buf.copy_and_advance(s.convert());
        }

        result.set_len(len);
        return result;
    }
}
```

CVE-2020-36323

A higher-order invariant bug in string join()

Found by Rudra in the Rust standard library

(Code simplified and renamed for presentation)

Bug Example

```
// join(&["a", "b", "c"], "|") => "a|b|c"
fn join<T>(array: &[T], sep: &str) -> String
    where T: Convert<str>
{
    // code that handles array.len() == 0 or 1
    ...

    let len = sep.len() * (array.len() - 1)
        + array.iter().map(|s| s.convert().len()).sum();

    let mut result = String::with_capacity(len);

    unsafe {
        // contains uninitialized bytes
        let mut buf = result.get_unchecked_mut(..len);

        buf.copy_and_advance(array[0].convert());
        for s in &array[1..] {
            buf.copy_and_advance(sep);
            buf.copy_and_advance(s.convert());
        }

        result.set_len(len);
        return result;
    }
}
```

CVE-2020-36323

A higher-order invariant bug in string join()

Found by Rudra in the Rust standard library

(Code simplified and renamed for presentation)

Bug Example

```rust
// join(&["a", "b", "c"], "|") => "a|b|c"
fn join<T>(array: &[T], sep: &str) -> String
    where T: Convert<str>
{
    // code that handles array.len() == 0 or 1
    ...

    let len = sep.len() * (array.len() - 1)
        + array.iter().map(|s| s.convert().len()).sum();

    let mut result = String::with_capacity(len);

    unsafe {
        // contains uninitialized bytes
        let mut buf = result.get_unchecked_mut(..len);

        buf.copy_and_advance(array[0].convert());
        for s in &array[1..] {
            buf.copy_and_advance(sep);
            buf.copy_and_advance(s.convert());
        }

        result.set_len(len);
        return result;
    }
}
```

```rust
// Custom convert implementation
fn convert(&self) -> &str {
    if self.first_time() {
        "123456"
    } else {
        "0"
    }
}
```

CVE-2020-36323

A higher-order invariant bug in string join()

Found by Rudra in the Rust standard library

(Code simplified and renamed for presentation)

Challenges

```rust
// join(&["a", "b", "c"], "|") => "a|b|c"
fn join<T>(array: &[T], sep: &str) -> String
    where T: Convert<str>
{
    // code that handles array.len() == 0 or 1
    ...

    let len = sep.len() * (array.len() - 1)
        + array.iter().map(|s| s.convert().len()).sum();

    let mut result = String::with_capacity(len);

    unsafe {
        // contains uninitialized bytes
        let mut buf = result.get_unchecked_mut(..len);

        buf.copy_and_advance(array[0].convert());
        for s in &array[1..] {
            buf.copy_and_advance(sep);
            buf.copy_and_advance(s.convert());
        }

        result.set_len(len);
        return result;
    }
}
```

Challenges

1. Incomplete definitions

```
// join(&["a", "b", "c"], "|") => "a|b|c"
fn join<T>(array: &[T], sep: &str) -> String
    where T: Convert<str>
{
    // code that handles array.len() == 0 or 1
    ...

    let len = sep.len() * (array.len() - 1)
        + array.iter().map(|s| s.convert().len()).sum();

    let mut result = String::with_capacity(len);

    unsafe {
        // contains uninitialized bytes
        let mut buf = result.get_unchecked_mut(..len);

        buf.copy_and_advance(array[0].convert());
        for s in &array[1..] {
            buf.copy_and_advance(sep);
            buf.copy_and_advance(s.convert());
        }

        result.set_len(len);
        return result;
    }
}
```

```
// join(&["a", "b", "c"], "|") => "a|b|c"
fn join<T>(array: &[T], sep: &str) -> String
    where T: Convert<str>
{
    // code that handles array.len() == 0 or 1
    ...

    let len = sep.len() * (array.len() - 1)
        + array.iter().map(|s| s.convert().len()).sum();

    let mut result = String::with_capacity(len);

    unsafe {
        // contains uninitialized bytes
        let mut buf = result.get_unchecked_mut(..len);

        buf.copy_and_advance(array[0].convert());
        for s in &array[1..] {
            buf.copy_and_advance(sep);
            buf.copy_and_advance(s.convert());
        }

        result.set_len(len);
        return result;
    }
}
```

Challenges

1. Incomplete definitions

2. Some information is not available in later compiler stages

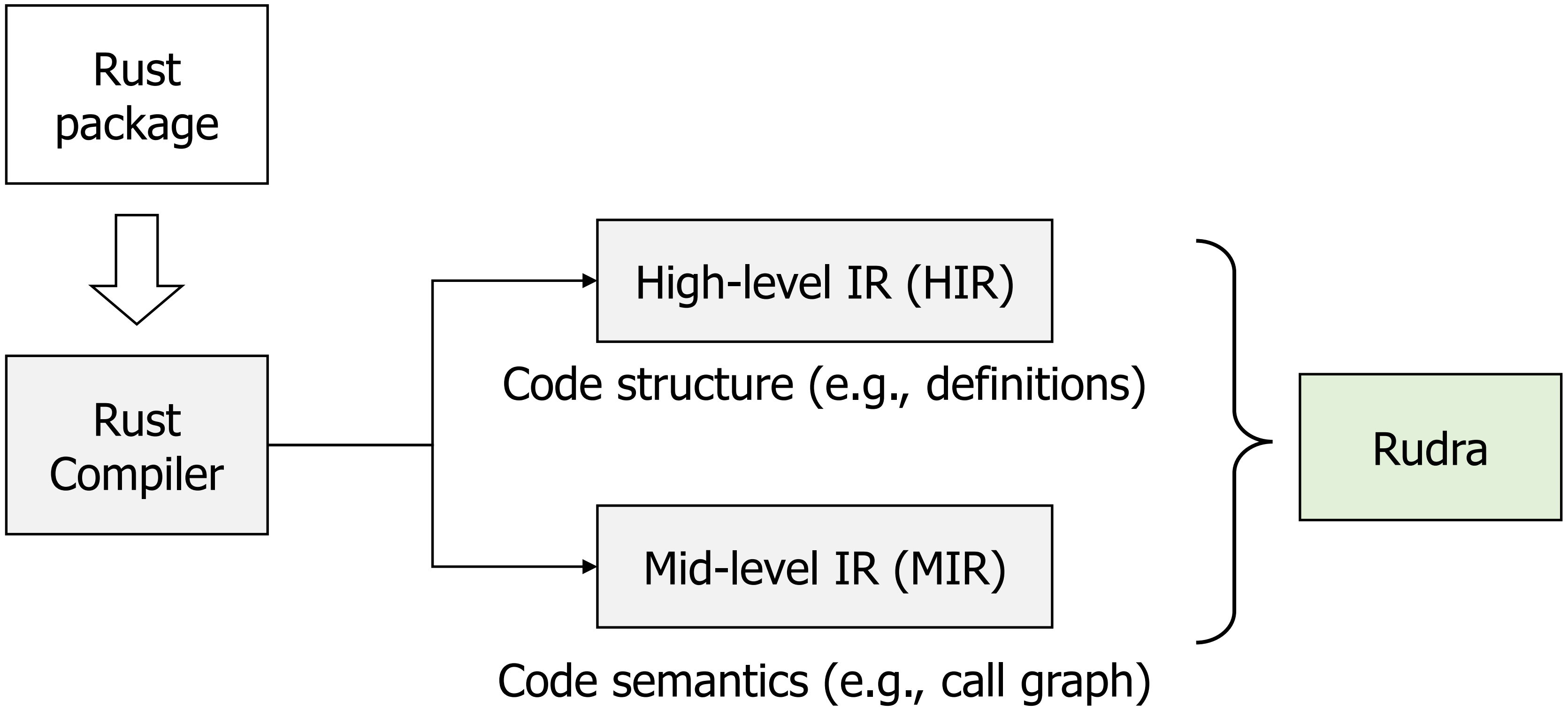
```rust
// join(&["a", "b", "c"], "|") => "a|b|c"
fn join<T>(array: &[T], sep: &str) -> String
    where T: Convert<str>
{
    // code that handles array.len() == 0 or 1
    ...

    let len = sep.len() * (array.len() - 1)
        + array.iter().map(|s| s.convert().len()).sum();

    let mut result = String::with_capacity(len);

    unsafe {
        // contains uninitialized bytes
        let mut buf = result.get_unchecked_mut(..len);

        buf.copy_and_advance(array[0].convert());
        for s in &array[1..] {
            buf.copy_and_advance(sep);
            buf.copy_and_advance(s.convert());
        }

        result.set_len(len);
        return result;
    }
}
```

Challenges

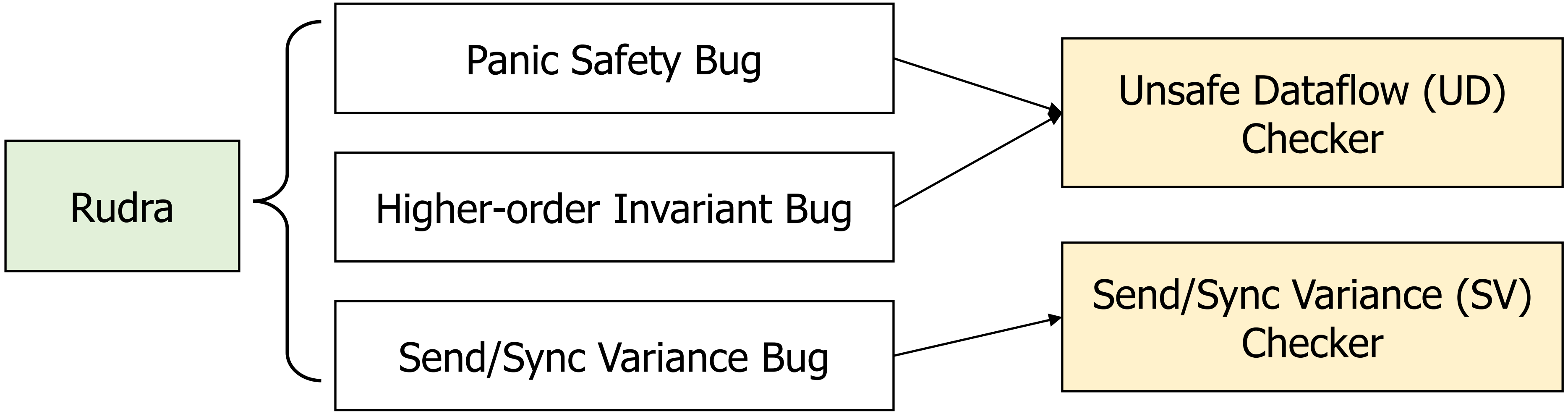
1. Incomplete definitions
2. Some information is not available in later compiler stages

➔ Flow-based heuristics that intermixes IRs at different compiler stages

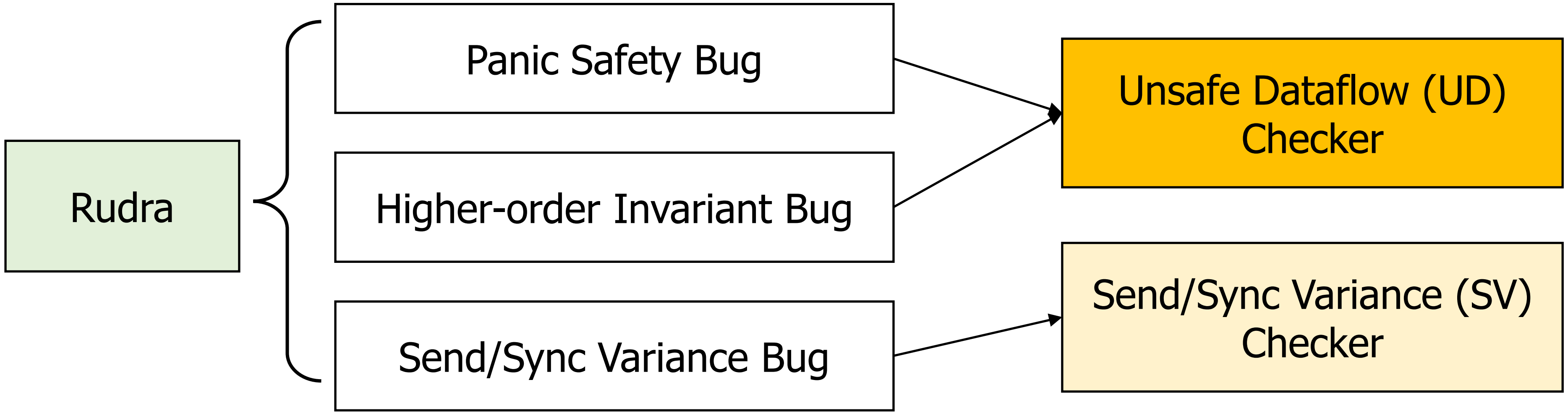
Rudra's Design



Rudra's Design



Rudra’s Design



- Uninitialized
- Ownership Duplicate
- Overwrite
- Buffer Copy
- Transmute
- Raw pointer conversion

Safety bypass ⟹ Implicit assumptions

UD Checker Example

```rust
// join(&["a", "b", "c"], "|") => "a|b|c"
fn join<T>(array: &[T], sep: &str) -> String
    where T: Convert<str>
{
    // code that handles array.len() == 0 or 1
    ...

    let len = sep.len() * (array.len() - 1)
        + array.iter().map(|s| s.convert().len()).sum();

    let mut result = String::with_capacity(len);

    unsafe {
        // contains uninitialized bytes
        let mut buf = result.get_unchecked_mut(..len);

        buf.copy_and_advance(array[0].convert());
        for s in &array[1..] {
            buf.copy_and_advance(sep);
            buf.copy_and_advance(s.convert());
        }

        result.set_len(len);
        return result;
    }
}
```

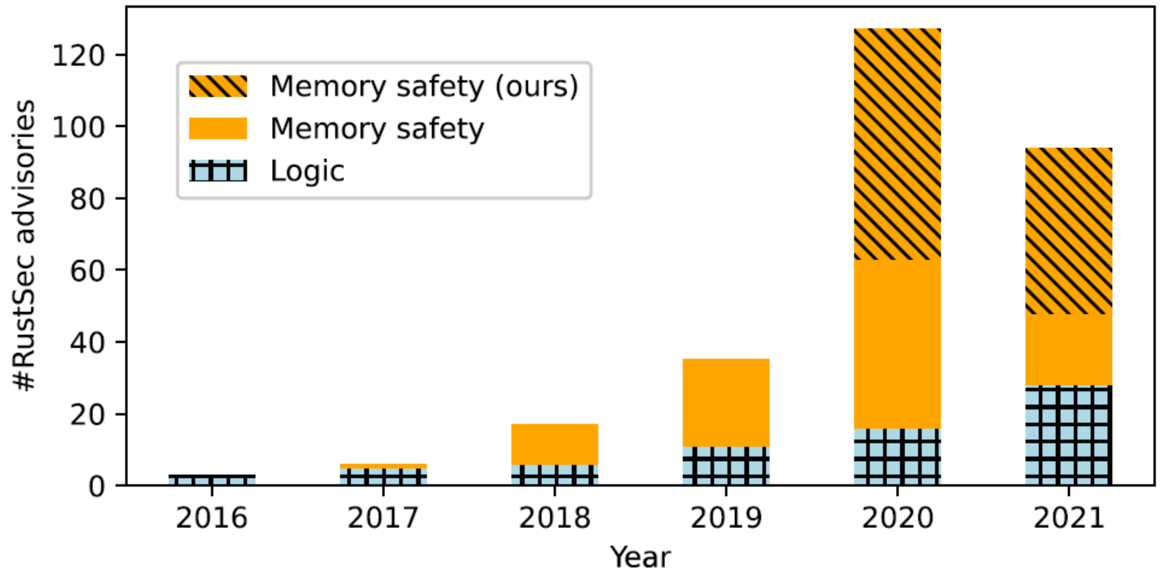
Safety bypass

⇩

Implicit assumptions

Evaluation: Bugs

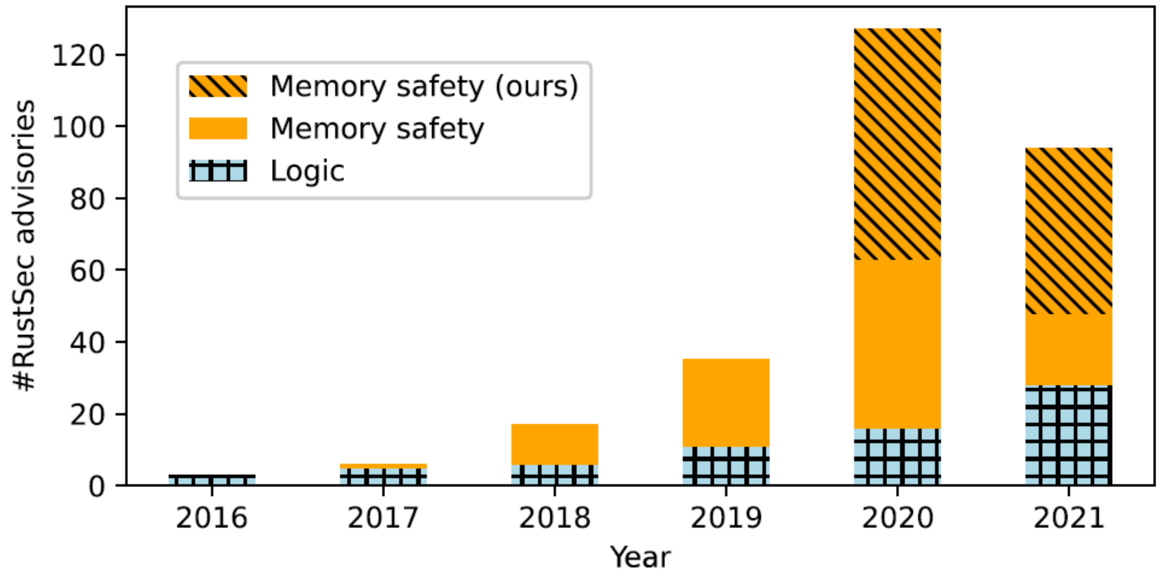
- Analyzed all 43k packages uploaded to Rust's main package repository
- 264 unknown memory safety bugs throughout the Rust ecosystem
  - 2 bugs in the Rust standard library
  - 1 bug in the official futures package for asynchronous programming
  - 1 design issue in the Rust compiler
  - 112 RustSec advisory
  - 76 CVEs



Evaluation: Bugs

- Analyzed all 43k packages uploaded to Rust's main package repository
- 264 unknown memory safety bugs throughout the Rust ecosystem
  - 2 bugs in the Rust standard library
  - 1 bug in the official futures package for asynchronous programming
  - 1 design issue in the Rust compiler
  - 112 RustSec advisory
  - 76 CVEs

➔ Rudra can find subtle and non-trivial bugs



Evaluation: Comparison

- Compared Rudra with dynamic analyzers: Fuzzers and Miri [1]
- Compared Rudra with a static analyzer: UAFChecker [2]
- **Result**
  - None of the bugs found by Rudra are detected by these methods
  - Miri found additional bugs not covered by Rudra's algorithms

[1] Ralf Jung, et al. **Stacked borrows: an aliasing model for Rust.**
Proceedings of the ACM on Programming Languages 4. POPL (2019): 1-32.

[2] Qin, Boqin, et al. **Understanding memory and thread safety practices and issues in real-world Rust programs.**
Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation. 2020.

Evaluation: Comparison

- Compared Rudra with dynamic analyzers: Fuzzers and Miri [1]
- Compared Rudra with a static analyzer: UAFChecker [2]
- **Result**
  - None of the bugs found by Rudra are detected by these methods
  - Miri found additional bugs not covered by Rudra's algorithms

➔ Rudra can find unique bugs

[1] Ralf Jung, et al. **Stacked borrows: an aliasing model for Rust.**
Proceedings of the ACM on Programming Languages 4. POPL (2019): 1-32.

[2] Qin, Boqin, et al. **Understanding memory and thread safety practices and issues in real-world Rust programs.**
Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation. 2020.

Limitation

1. **Not exhaustive**

2. **False positive rate**
   - Around 50% at high precision mode, 80% at low precision mode

3. **Bugs are found at the definition site**

Rudra: A Static Analyzer for Unsafe Rust

- We identified three common bug patterns in unsafe Rust
- We devised two new algorithms to detect them
- We implemented a static analyzer named Rudra,
  that can scale to the entire Rust ecosystem (43k packages / 6.5 hours)
- Found more than half of the memory safety bugs
  known to the Rust security advisory database (RustSec)
  - 76 CVEs and 112 RustSec advisories
  - Including two memory safety bugs in the Rust standard library

https://github.com/sslab-gatech/Rudra