Building Bridges: Safe Interactions With Foreign Languages Through Omniglot

Muhammad Hassnain

Literature Review

August 1, 2025

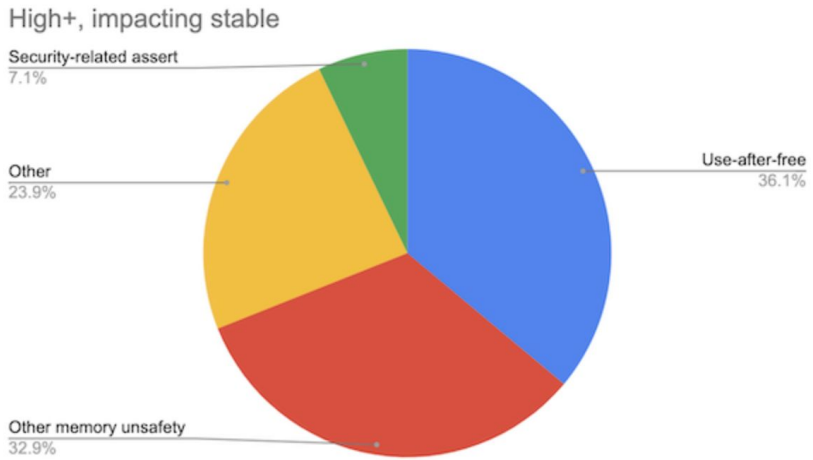
Chromium > Chromium Security >

Memory safety

The Chromium project finds that around 70% of our serious security bugs are memory safety problems. Our next major project is to prevent such bugs at source.

The problem

Around 70% of our high severity security bugs are memory unsafety problems (that is, mistakes with C/C++ pointers). Half of those are use-after-free bugs.



(Analysis based on 912 high or critical severity security bugs since 2015, affecting the Stable channel.)



SIGN IN / UP

The Register

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."

Image Credits : https://www.usenix.org/sites/default/files/conference/protected-files/osdi25_slides_schuermann_leon.pdf



The Linux Kernel

6.9.0-rc5

Contents

Development process
Submitting patches
Code of conduct
Maintainer handbook
All development-process docs

Core API
Driver APIs
Subsystems
Locking

Licensing rules

Rust

English

Documentation related to Rust within the kernel. To start using Rust in the kernel, please read the Quick Start guide.

The Rust experiment

The Rust support was merged in v6.1 into mainline in order to help in determining whether Rust as a language was suitable for the kernel, i.e. worth the tradeoffs.

Currently, the Rust support is primarily intended for kernel developers and maintainers interested in the Rust support, so that they can start working on abstractions and drivers, as well as helping the development of infrastructure and tools.

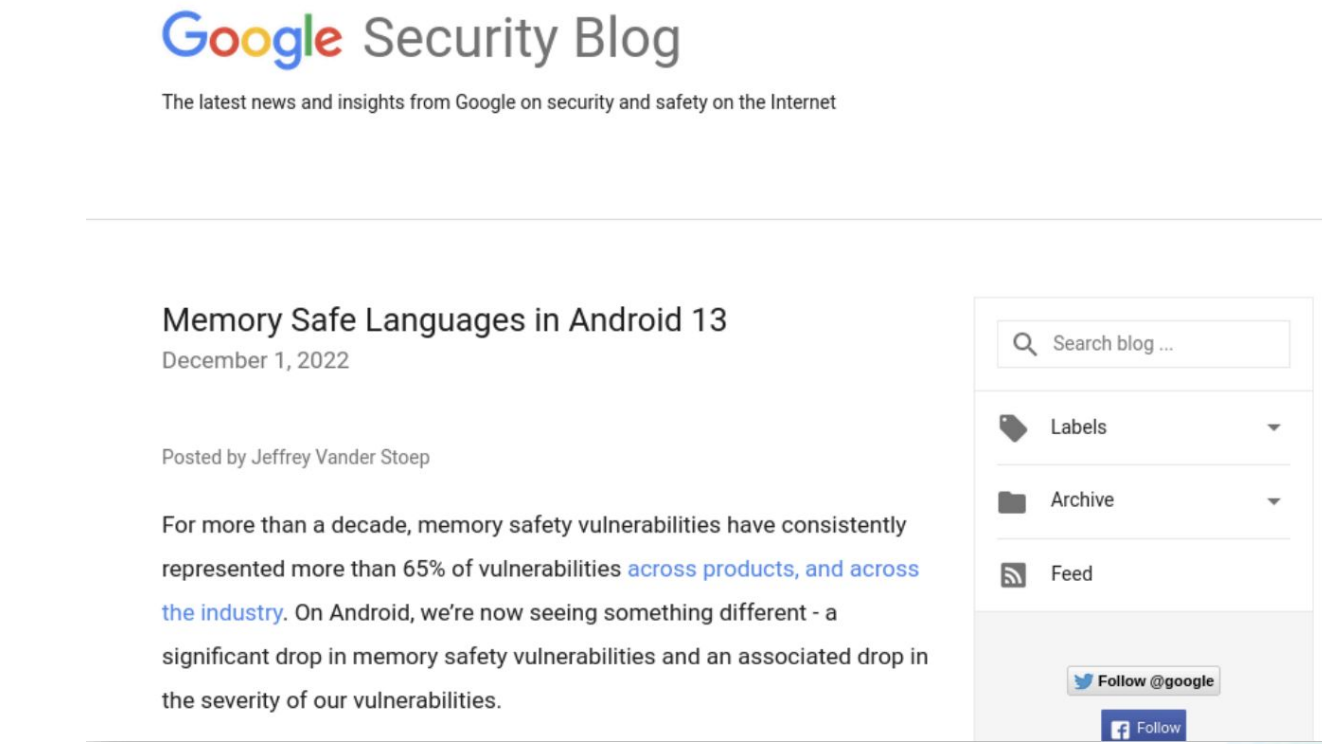
If you are an end user, please note that there are currently no in-tree drivers/modules suitable or intended for production use, and that the Rust support is still in development/experimental, especially for certain kernel configurations.

This documentation does not include rustdoc generated information.

- Quick Start
- General Information
- Coding Guidelines
- Arch Support
- Testing

©The kernel development community. | Powered by Sphinx 5.0.1 & Alabaster 0.7.12 | Page source

Image Credits : https://www.usenix.org/sites/default/files/conference/protected-files/osdi25_slides_schuermann_leon.pdf



Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Memory Safe Languages in Android 13

December 1, 2022

Posted by Jeffrey Vander Stoep

For more than a decade, memory safety vulnerabilities have consistently represented more than 65% of vulnerabilities across products, and across the industry. On Android, we're now seeing something different - a significant drop in memory safety vulnerabilities and an associated drop in the severity of our vulnerabilities.

Search blog ...

Labels

Archive

Feed

Follow @google

Follow

Image Credits : https://www.usenix.org/sites/default/files/conference/protected-files/osdi25_slides_schuermann_leon.pdf

Problems

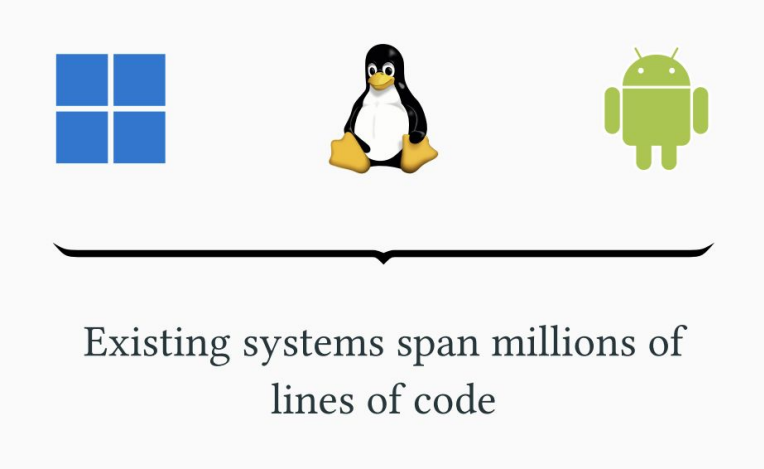


Image Credits : https://www.usenix.org/sites/default/files/conference/protected-files/osdi25_slides_schuermann_leon.pdf

Problems

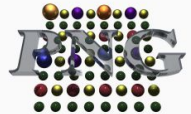






Existing systems span millions of lines of code



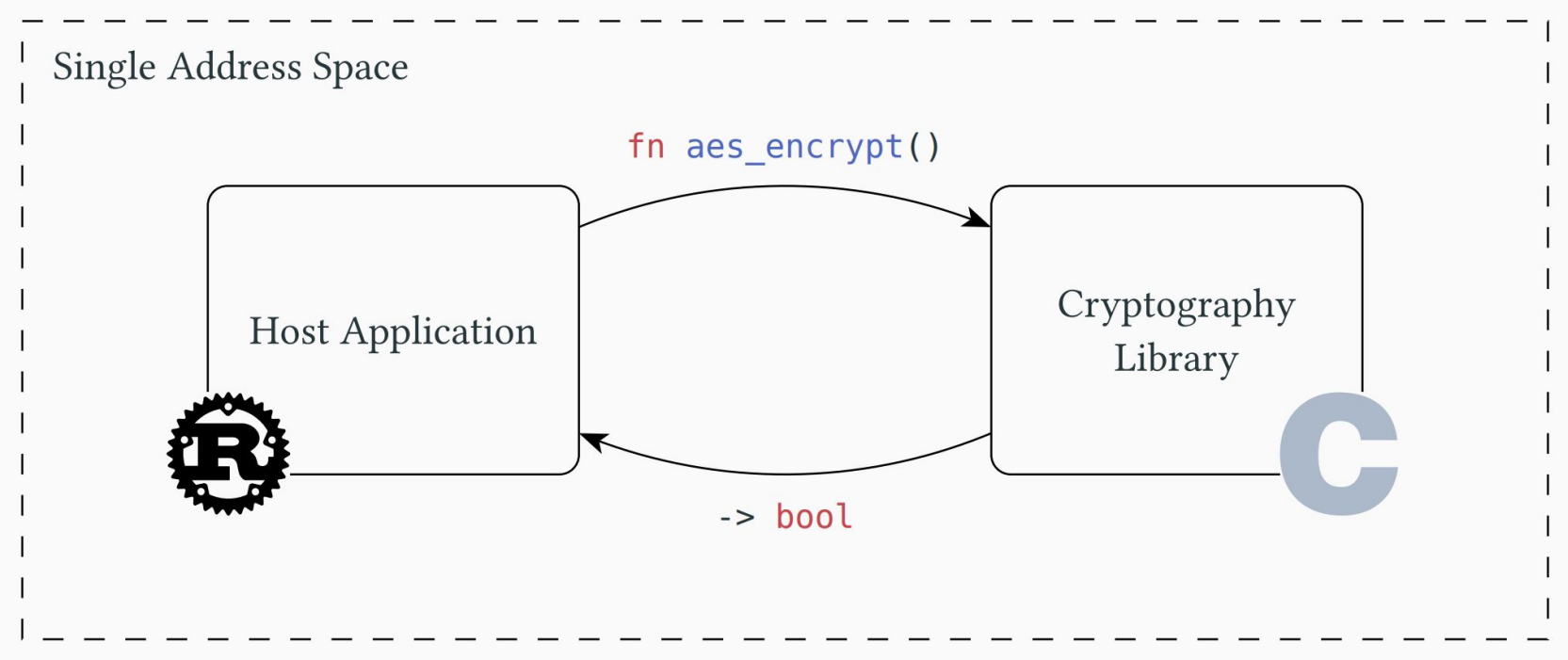


Even new systems must integrate existing, proven, optimized, and tested libraries

Rewriting everything is impractical!!

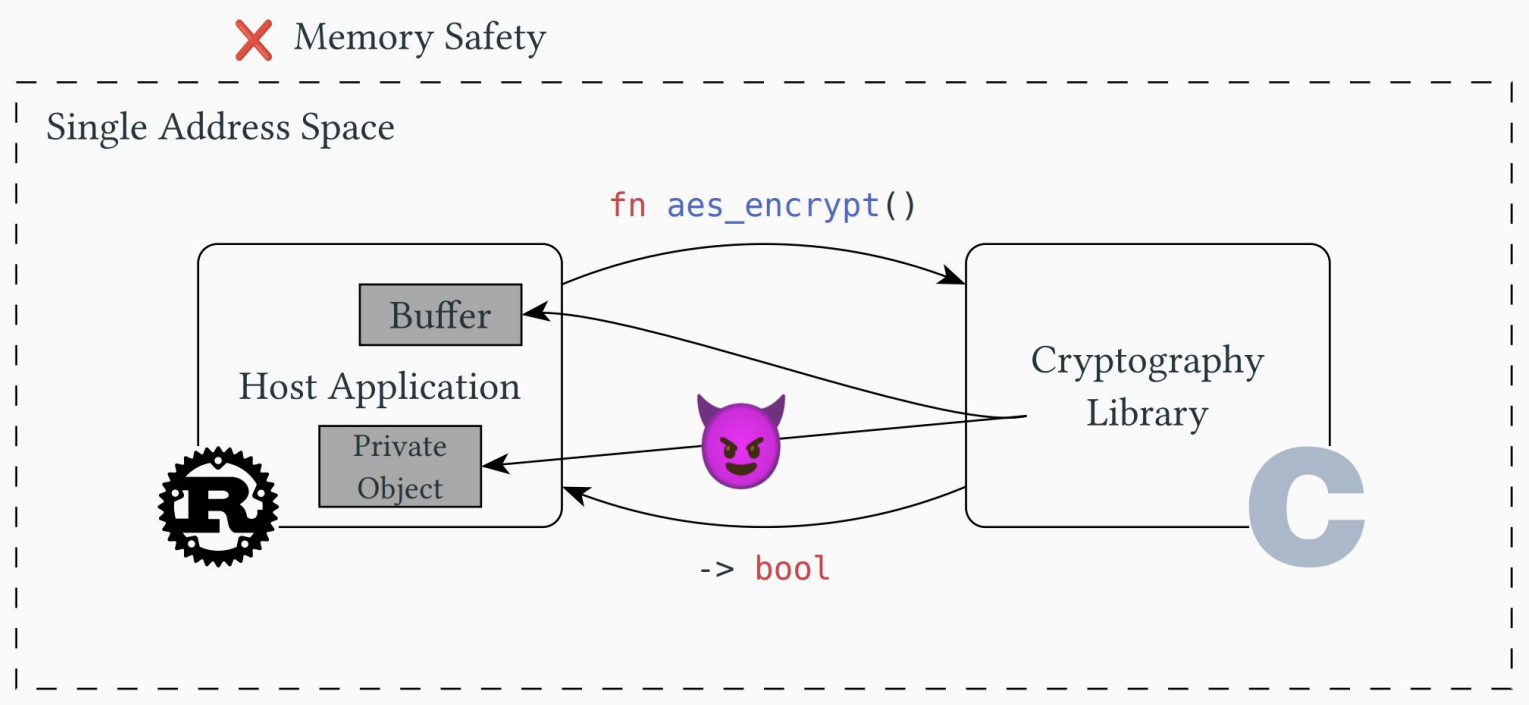
Image Credits : https://www.usenix.org/sites/default/files/conference/protected-files/osdi25_slides_schuermann_leon.pdf

Safe Language Interactions with Unsafe Code



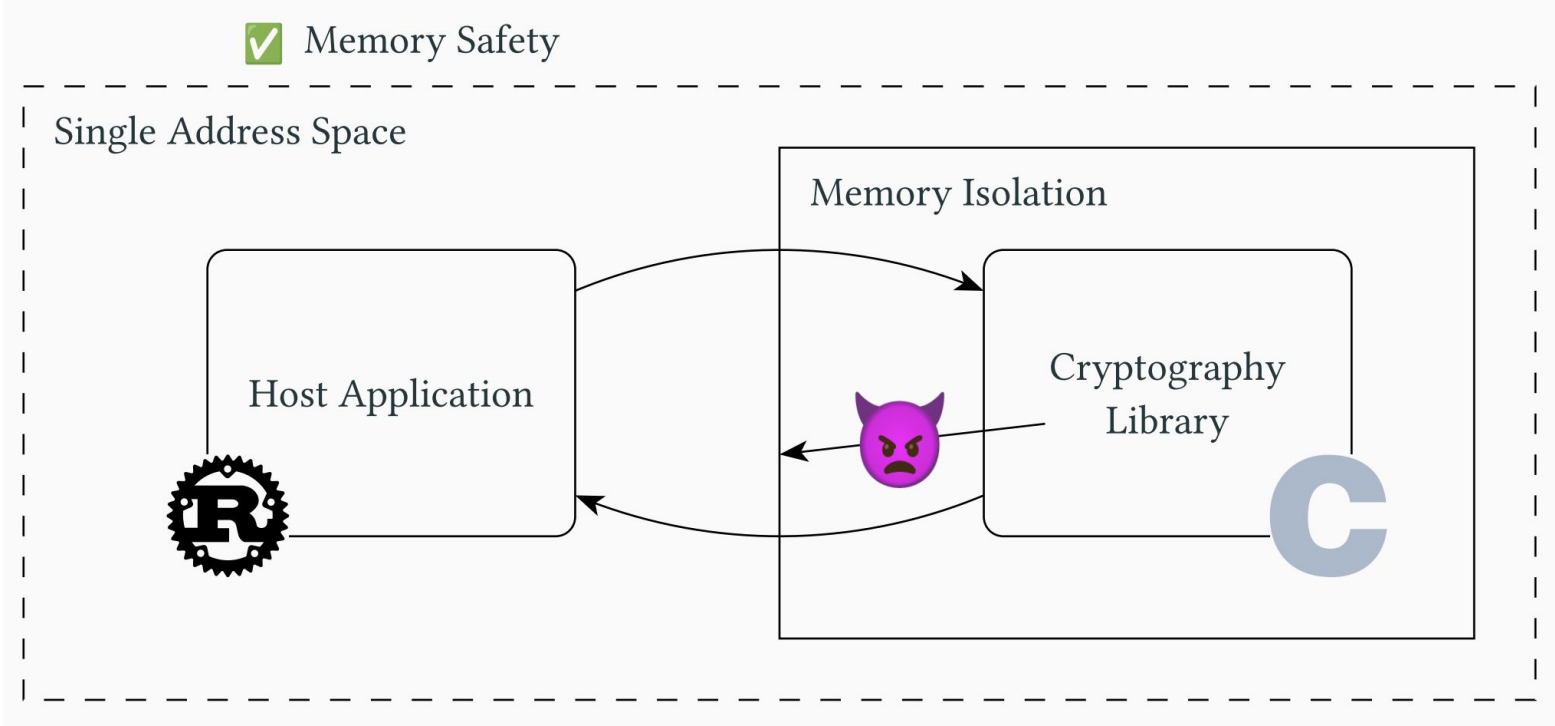
Slide Credits : https://www.usenix.org/sites/default/files/conference/protected-files/osdi25_slides_schuermann_leon.pdf

Safe Language Interactions with Unsafe Code



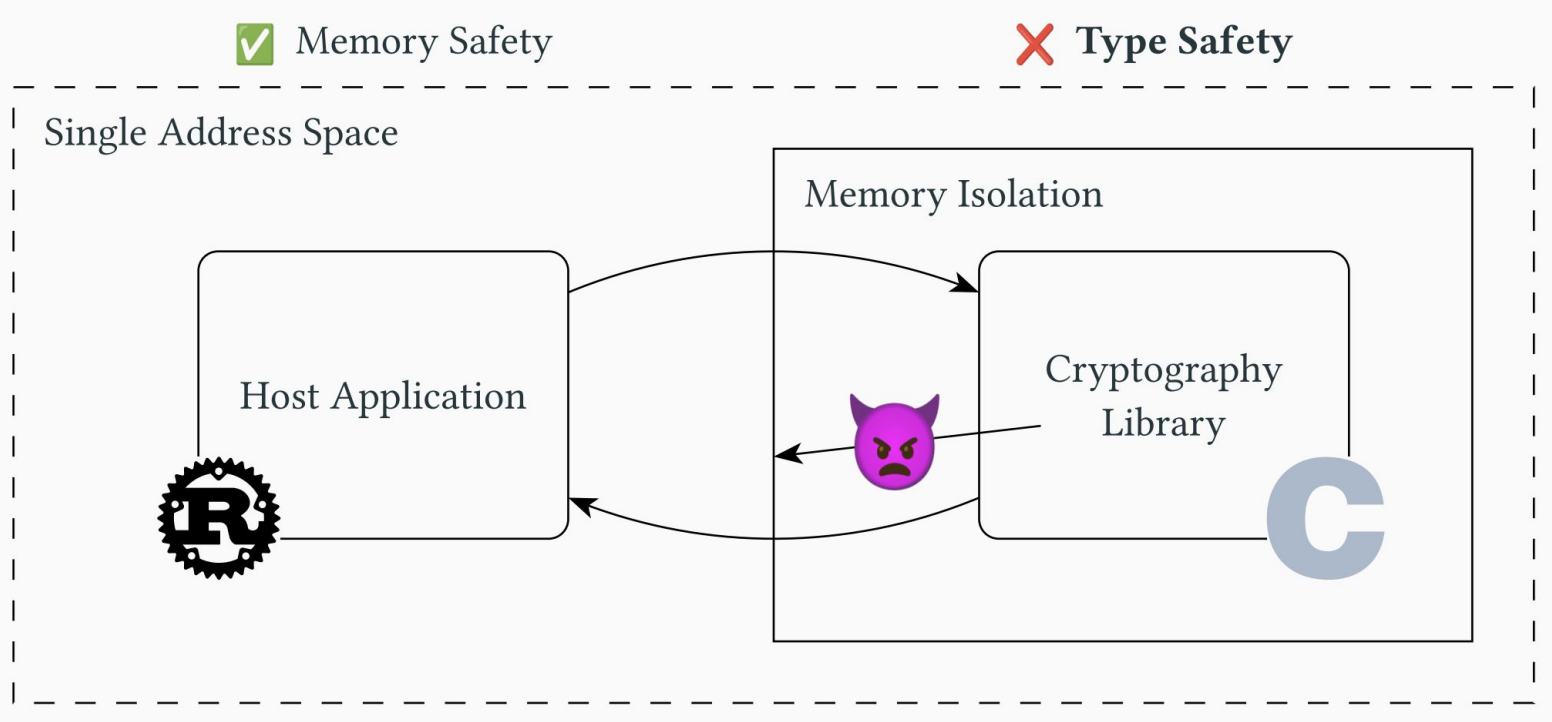
Slide Credits : https://www.usenix.org/sites/default/files/conference/protected-files/osdi25_slides_schuermann_leon.pdf

Safe Language Interactions with Unsafe Code



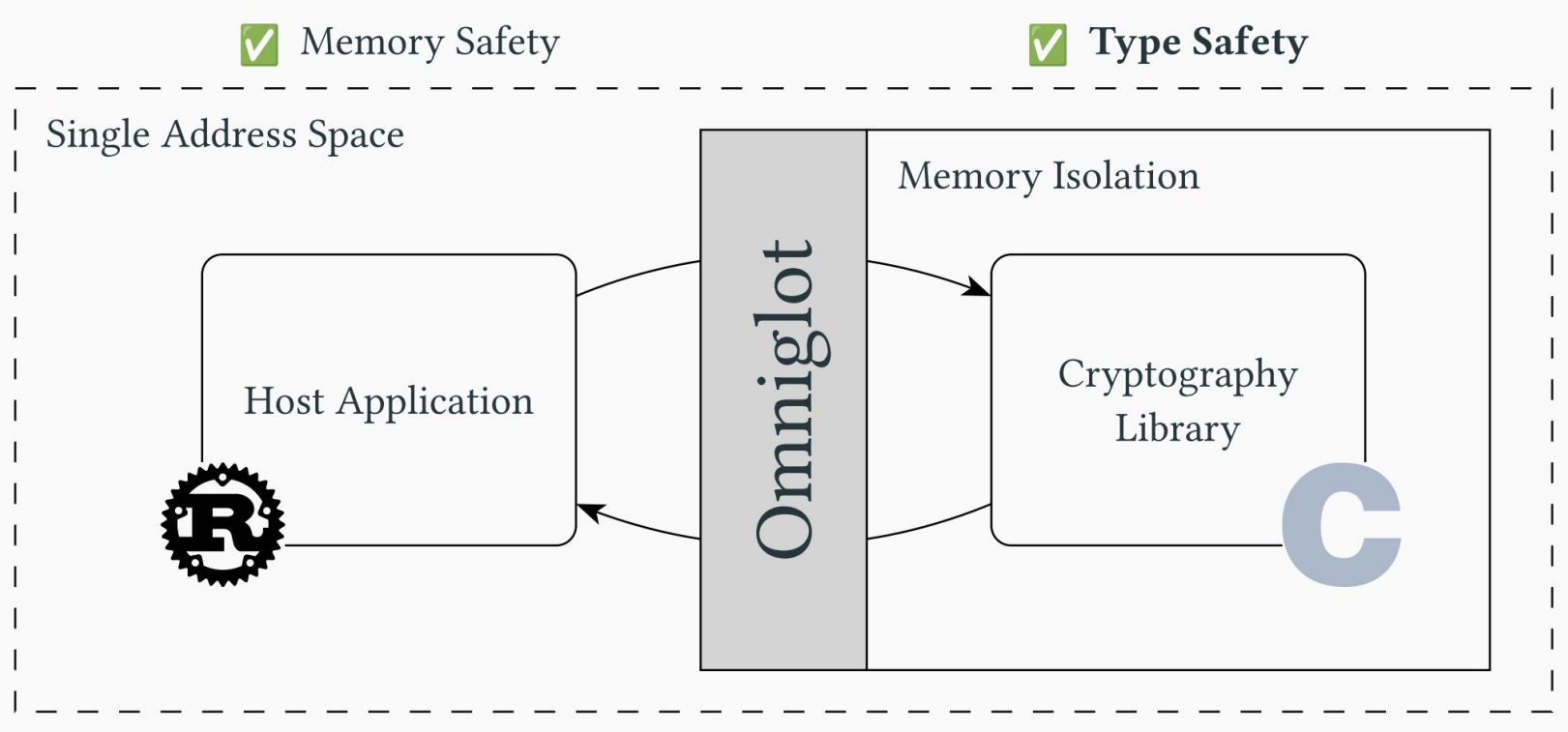
Slide Credits : https://www.usenix.org/sites/default/files/conference/protected-files/osdi25_slides_schuermann_leon.pdf

Safe Language Interactions with Unsafe Code



Slide Credits : https://www.usenix.org/sites/default/files/conference/protected-files/osdi25_slides_schuermann_leon.pdf

Safe Language Interactions with Unsafe Code



Slide Credits : https://www.usenix.org/sites/default/files/conference/protected-files/osdi25_slides_schuermann_leon.pdf

Omniglot

Omniglot mediates interactions between Rust and C code:

- Validating data types
- Preventing mutable aliasing

Type Violation can break memory safety

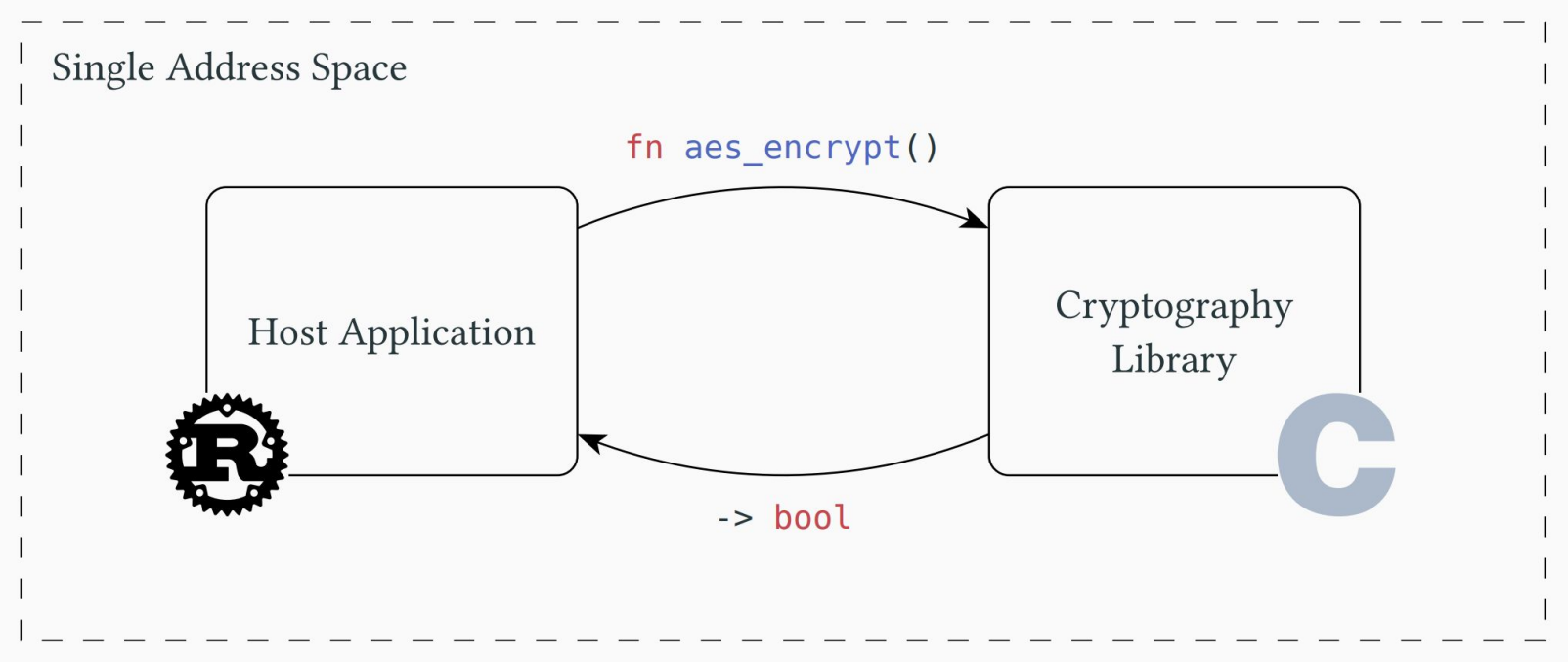
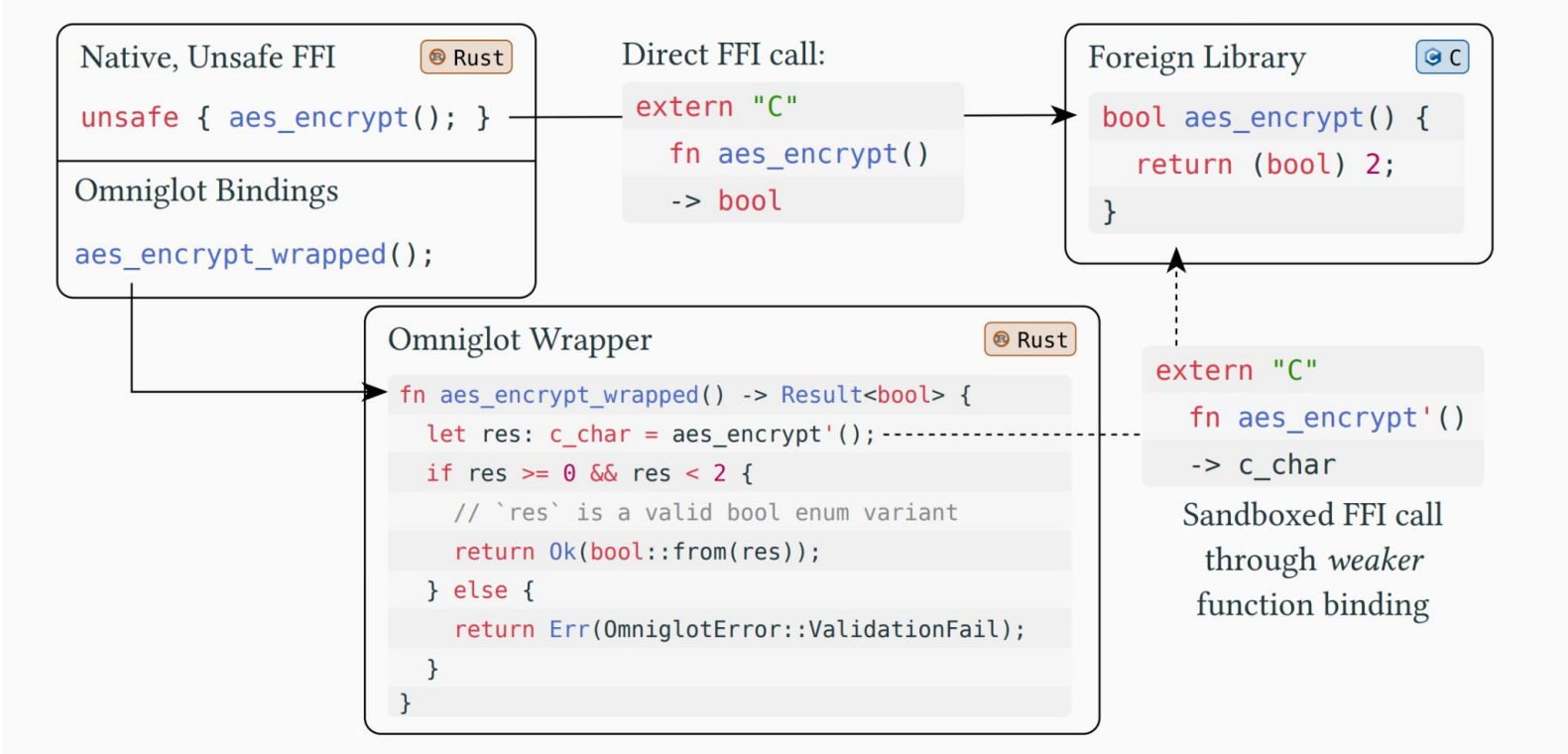


Image Credits : https://www.usenix.org/sites/default/files/conference/protected-files/osdi25_slides_schuermann_leon.pdf

What does Omniglot do?



Slide Credits : https://www.usenix.org/sites/default/files/conference/protected-files/osdi25_slides_schuermann_leon.pdf

Mutable Aliasing

Can you spot the problem!

```
1  pub enum Message<'a> {
2    Encrypted(bool, &'a [u8]),
3    Unencrypted(CString),
4  }
5
6  pub fn enc(src: &mut Vec<u8>) -> Message<'_> {
7    let mut dst: *const u8 = ptr::null();
8    unsafe {
9      // Calling aes_encrypt from Listing 1
10     let res = aes_encrypt(KEY, src.as_mut_ptr(),
11       &mut dst, src.len(), IN_PLACE);
12     Message::Encrypted(res,
13       slice::from_raw_parts(dst, src.len()))
14   }
15 }
```

Image Source : https://www.usenix.org/system/files/osdi25-schuermann.pdf

Omniglot pointer model

1. *mut T
    - This is a raw pointer with no invariants.
    - Can be dangling, misaligned and alias host memory.
    - Safe Rust cannot dereference it.

Omniglot pointer model

2. OGMutRef<T>

   - Points to Active, Well Aligned, Foreign Memory of sufficient size for Type T.
   - Can be obtained by upgrading the raw pointer at runtime.
   - Writing to it does not violate memory safety.

Omniglot pointer model

2. OGMutRef<T>

This is essentially an UnsafeCell<MaybeUninit<T>>

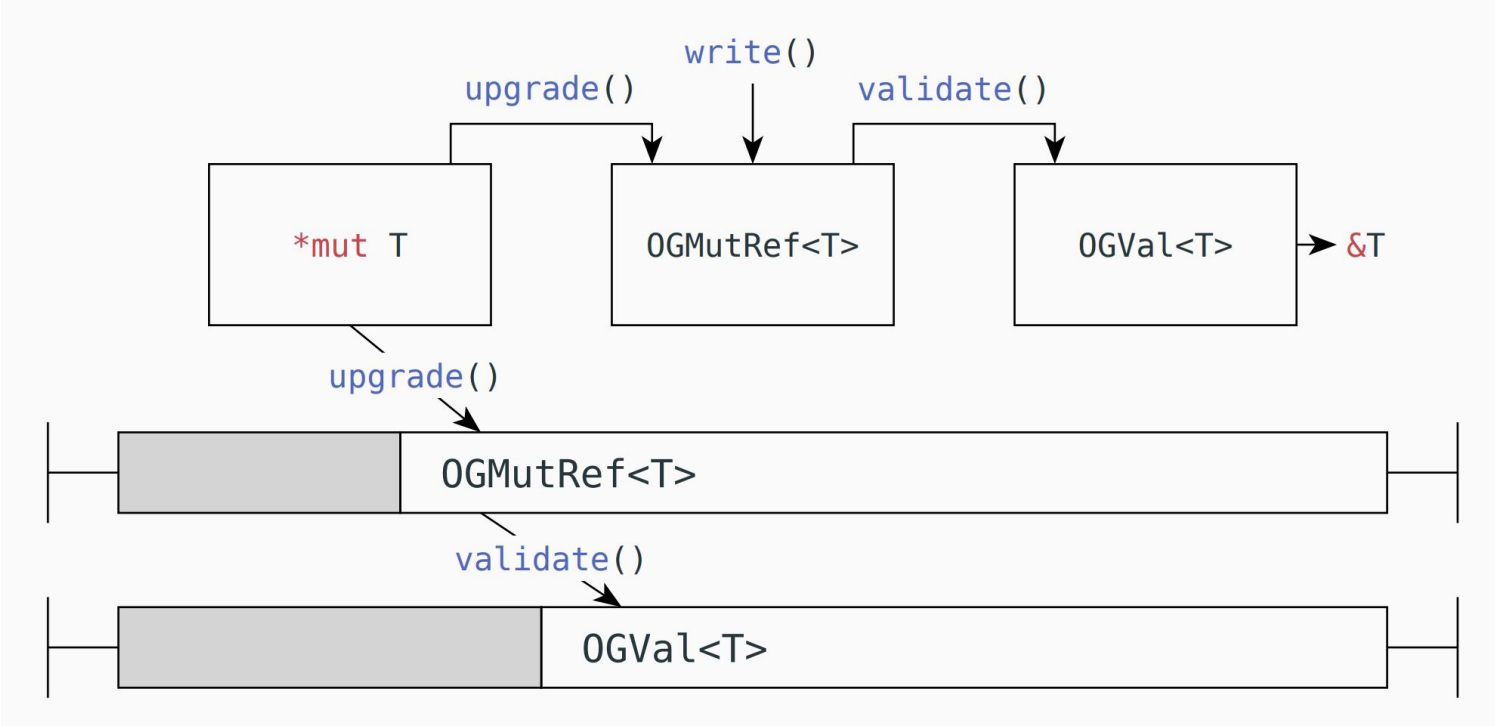
- MaybeUninit → stops Rust compiler from making any assumptions about the validity of values behind the pointer.
- Unsafe Cell → allows values to be mutable aliased. (can only be used if all other references to this location has same annotation).

Omniglot pointer model

3. OGVal<T>

   - Refers to memory which contains an actual, dereferencable instance of type T.
   - Can be obtained by validating OGMutRef<T> by inspecting underlying bit pattern.

Omniglot pointer model



Slide Credits : https://www.usenix.org/sites/default/files/conference/protected-files/osdi25_slides_schuermann_leon.pdf

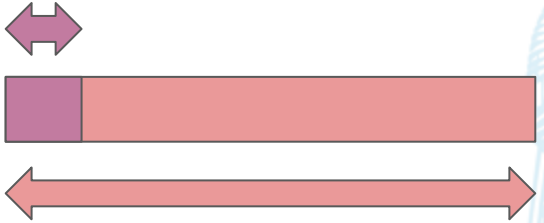
Things that can go wrong

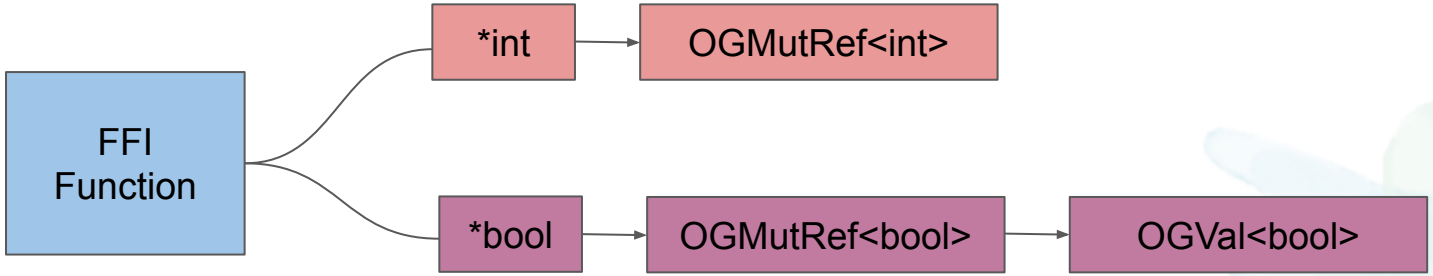
→ Writing to OGMutRef<T> which was previously validated as OGVal<T>

→ Underlying Foreign Memory may be revoked. (Dangling, UAF etc)

→ Invoking a FFI may modify memory.

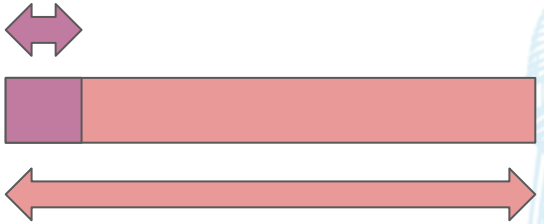
Question

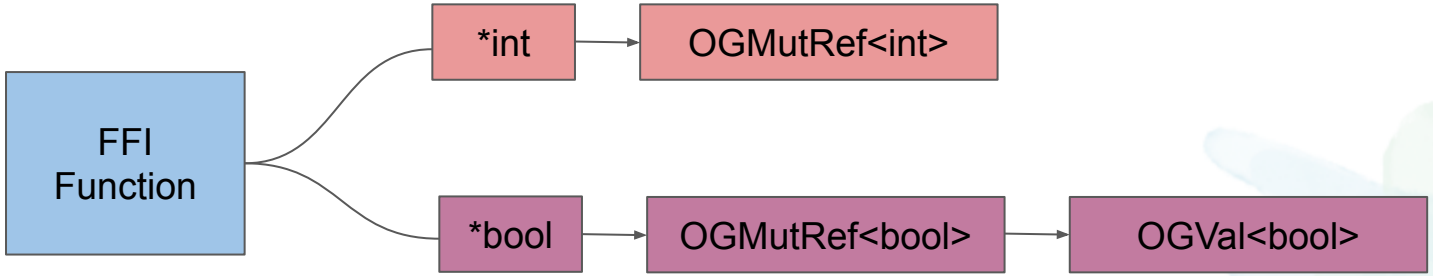




- Is writing to OGMutRef<int> safe?
- How does it impact OGVal<bool>?

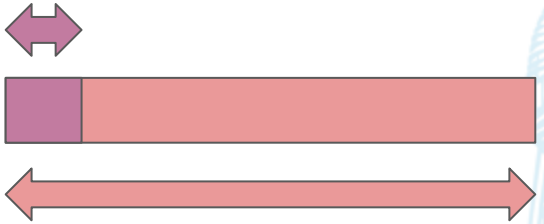
Fix

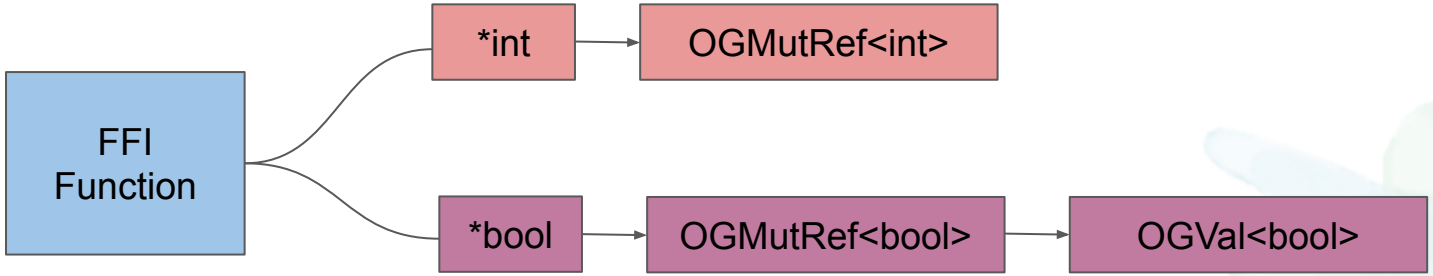




Creating an OGVal<T> acquires a read lock internally and it is released when OGVal goes out of scope. Similarly invoking OGMutRef::write() acquires a write lock.

Question





Since, they enforce these locks to protect against allocation and revocation, they should still be vulnerable if FFI library calls free?

Omniglot API Walkthrough

Image Source : https://www.usenix.org/system/files/osdi25-schuermann.pdf

```
1       // Load a library into a runtime, creating
2       // allocation and access scope markers:
3   let (rt, lib, mut oalloc, mut access) =
4     Rt::new("libcompress.so");
5
6   // Allocate a [u8; 4] on the foreign stack
7   rt.stack_alloc_t::<[u8; 4]>(
8     &mut oalloc, |array, ialloc| {
9       // array: OGMutRef<'_, [u8; 4]>
10      array.write([0, 1, 2, 3], &mut access);
11
12      let validated = array.validate(&access);
13      println!("{:?}", validated);
14
15      let compressed_len = lib.compress(
16        buf: array.as_ptr(),
17        length: 4,
18        &mut ialloc, &mut access).validate();
19
20      // Would not compile, as `validated` is
21      // bound to the previous access scope:
22      println!("{:?}", validated);
23
24      let upgraded = array.as_ptr()
25        .upgrade_slice(compressed_len, &ialloc);
26      let revalidated = upgraded.validate(&access);
27      println!("{:?}", revalidated);
28
29      // `array` and `upgraded` cannot escape
30      // closure, capture the `ialloc` scope.
31    }
32  )
33  Active Access Scope
34  Active Allocation Scope
```

`OGRt::new()` yields a unique allocation and access scope per library: `OGRt::new(library) -> (Rt, AllocScope, AccessScope)`

Allocation scopes prevent dangling references:
`ptr::upgrade(&AllocScope<'l>) -> OGMutRef<'l>`
`OGRt::stack_alloc(&'a mut AllocScope<'l>, Fn(&'b mut AllocScope<'m>))`

Access scopes prevent invalidating references through writes:
`OGMutRef::write(val: T, &mut AccessScope)`
`OGMutRef::validate(&'a AccessScope) -> OGVal<'a>`

`invoke` may write to memory and modify allocations in callbacks: `OGRt::invoke(&mut AllocScope, &mut AccessScope)`

Runtimes

→ Tock Operating System with RISC-V Physical Memory Protection(PMP)

→ Linux Userspace (using x86 memory protection keys)

Tock Operating System

→ Tock kernel written in Rust and is used for security critical applications.

→ PMP loads foreign libraries into ‘protection domains’ ⇒ they run at a lower privilege level and do not have access to foreign memory.

→ No syscalls interface accessible to foreign library.

→ No concurrency primitives as well.

Linux Userspace

→ Uses MPK keys to enforce ‘isolation’

→ Foreign library is not allowed to access host or other libraries memory.

→ They overwrite malloc and free to enforce memory isolation aware allocation.

→ Syscalls are not protected , hence developers must manually check that to protect against concurrency.

⇒ This paper does not address the inherent limitations of MPK, so this is a weak runtime for Omniglot.

Evaluation

Library	Runtime	#invoke	#callback	#upgrade	#validate	unsafe	isolation only		Omniglot	
`CryptoLib`	PMP	14	0	10	2	9105 µs	9145 µs	(+0.44%)	9145 µs	(+0%)
`LittleFS`	PMP	7	0	0	7	3115.3 µs	3742.8 µs	(+20.1%)	3764 µs	(+0.5%)
`LwIP`	PMP	3	1	2	5	51.74 µs	78.71 µs	(+52.1%)	81.4 µs	(+3.4%)
`Brotli`	MPK	2	0	0	4	3.12 ms	3.14 ms	(+0.6%)	3.14 ms	(+0%)
`libsodium`	MPK	1	0	0	2	51.61 µs	53.41 µs	(+3.4%)	53.41 µs	(+0%)
`libpng`	MPK	5	15	16	3	352.98 µs	397.93 µs	(+12.7%)	401.25 µs	(+0.8%)

Table 2: Omniglot overheads for a diverse set of libraries, across two runtimes. We show the number of Omniglot operations per benchmark iteration, and compare its overheads to an unsafe baseline, and memory isolation without Omniglot's runtime checks.

Image Source : https://www.usenix.org/system/files/osdi25-schuermann.pdf

Evaluation

[Sandcrust: Automatic Sandboxing of Unsafe Components in Rust]

^this is from 2017

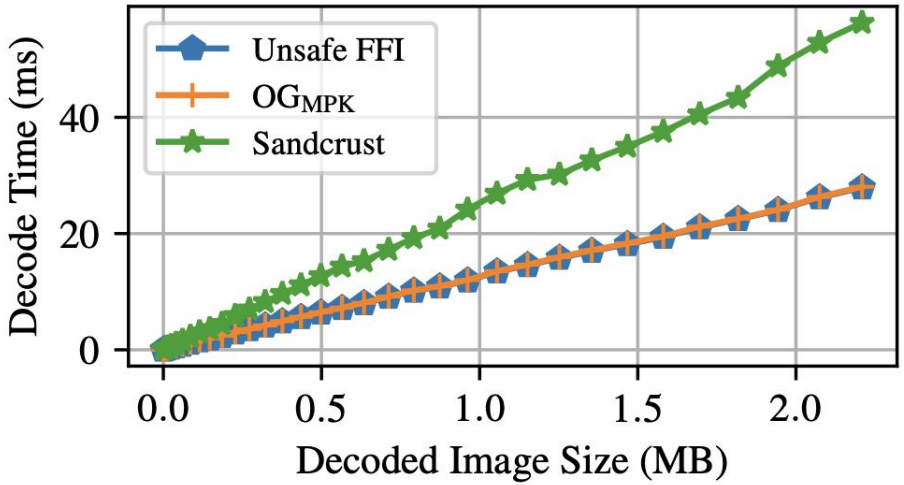


Figure 5: Execution time of decoding a PNG image, as function of the decoded image size. We compare OG$_{MPK}$ against an unsafe FFI baseline and Sandcrust. Omniglot's ability to perform zero-copy accesses to foreign memory allows it to outperform Sandcrust, which requires serialization and copying.

Image Source : https://www.usenix.org/system/files/osdi25-schuermann.pdf

Evaluation

Sub-table (c) validation time is linear with respect to the size of data validated.

→ If we use a SDB or Page Table based approach we should be faster??

(a)	Setup		Invoke	
	PMP	MPK	PMP	MPK
Unsafe	0.17 µs	1.49 ms	0.17 µs	13.74 ns
Omniglot	73.88 µs	4.08 ms	6.57 µs	98.90 ns
Sandcrust		1.79 ms		10.87 µs
Tock Upcall			56.20 µs	

(b)	Upgrade		Callback	
allocs / CBs	PMP	MPK	PMP	MPK
1	0.6 µs	32.30 ns	8.66 µs	230.1 ns
8	1.7 µs	164.1 ns	9.82 µs	364.3 ns
64	9.4 µs	1.10 µs	18.22 µs	1.32 µs

(c) Validate	64 B `u8`	8 kB `u8`	64 B `str`	8 kB `str`
PMP	0.22 µs	0.23 µs	2.74 µs	161.5 µs
MPK	1.67 ns	1.70 ns	359.6 ns	70.94 µs

Table 3: Omniglot's overheads broken down by operation.

Image Source : https://www.usenix.org/system/files/osdi25-schuermann.pdf

Discussion/Questions