Code Migration & Memory Safety

CRESP Industry Day, November 14, 2024

Per Larsen, Immunant, Inc.

Who?

- Immunant, Inc.
  - Code migration from C to Rust
  - Prevention of memory corruption vulnerabilities
  - Rust training 🦀
- Per
  - CEO at Immunant, Inc.
  - Postdoc at University of California, Irvine
  - PhD from Technical University of Denmark 🇩🇰

BAD PRACTICES

https://www.cisa.gov/resources-tools/resources/product-security-bad-practices

BAD PRACTICES

Development in Memory Unsafe Languages (CWE[1]-119 and related weaknesses)

The development of new product lines for use in service of critical infrastructure or NCFs in a memory-unsafe language (e.g., C or C++) where there are readily available alternative memory-safe languages that could be used is dangerous and significantly elevates risk to national security, national economic security, and national public health and safety.

For existing products that are written in memory-unsafe languages, not having a published memory safety roadmap by January 1, 2026 is dangerous and significantly elevates risk to national security, national economic security, and national public health and safety. The memory safety roadmap should outline the manufacturer's prioritized approach to eliminating memory safety vulnerabilities in priority code components (e.g., network-facing code or code that handles sensitive functions like cryptographic operations). Manufacturers should demonstrate that the memory safety roadmap will lead to a significant, prioritized reduction of memory safety vulnerabilities in the manufacturer's products and demonstrate they are making a reasonable effort to follow the memory safety roadmap.

Memory safety

- Means that software or a programming language is designed to catch memory bugs

Memory safety helps avoid

- Security Vulnerabilities
  - 60-90 percent of software flaws in memory unsafe languages
  - High fraction of zero-day vulnerabilities
- Program Instability
- Difficult Debugging and Maintenance

Can we make existing C/C++ code memory safe?

In principle, yes. In practice…

Can we make existing C/C++ code memory safe?

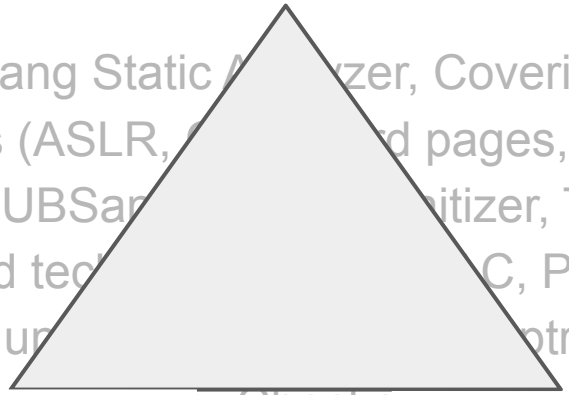
Work to make C/C++ safe includes

- Static analysis (Clang Static Analyzer, Coverity, CppCheck, …)
- Exploit mitigations (ASLR, CFI, guard pages, allocator hardening, …)
- Sanitizers (ASan, UBSan, MemorySanitizer, ThreadSanitizer, GWPAsan, …)
- Hardware-assisted techniques (MTE, PAC, PKRU, MPX, CET, CHERI, …)
- Modern C++ (std::unique_ptr, std::shared_ptr, std::weak_ptr, …)
- Language dialects (Cyclone, CheckedC, CCured)

Can we make existing C/C++ code memory safe?

Work to make C/C++ sa

- Static analysis (Clang Static A zer, Coverity, CppCheck, …)
- Exploit mitigations (ASLR, d pages, allocator hardening, …)
- Sanitizers (ASan, UBSa itizer, ThreadSanitizer, GWPAsan, …)
- Hardware-assisted te C, PKRU, MPX, CET, CHERI, …)
- Modern C++ (std::u tr, std::weak_ptr, …)
- Lan , Checke



Can we make existing C/C++ code memory safe?

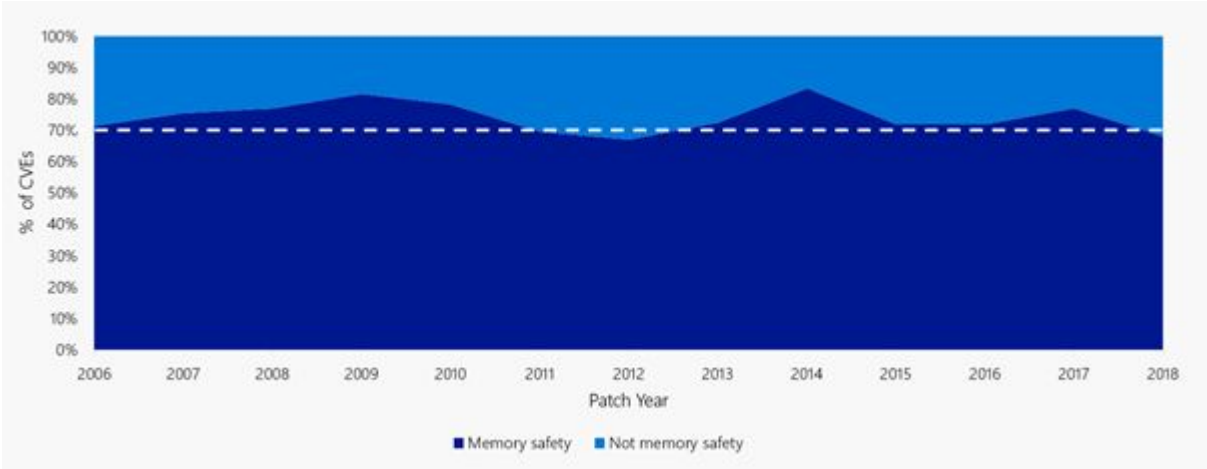
Work to make C/C++ safe includes

- Static analysis (Clang Static Ana… CppCheck, …)
- Exploit mitigations (AS… hardening, …)
- Sanitizers (A… nitizer, GWPAsan, …)
- Hardware-a… …, MPX, CET, CHERI, …)
- Modern C++ … ed_ptr, std::weak_ptr, …)
- Language dia… …, CheckedC, CCured)

IS IT WORTH THE HASSLE?

Can we make existing C/C++ code memory safe?

Microsoft: 70% of all vulnerabilities in their products over the last decade have been memory safety issues



https://msrc.microsoft.com/blog/2019/07/we-need-a-safer-systems-programming-language

Can we make existing C/C++ code memory safe?

- Google: 90% of Android vulnerabilities are memory safety issues
- Apple: 60-70% of vulnerabilities in iOS and macOS are memory safety vulnerabilities
- Alex Gaynor: more than 80% of the exploited zero-day vulnerabilities were memory safety issues

https://security.googleblog.com/2019/05/queue-hardening-enhancements.htm
https://langui.sh/2019/07/23/apple-memory-safety/
https://twitter.com/LazyFishBarrel/status/1129000965741404160

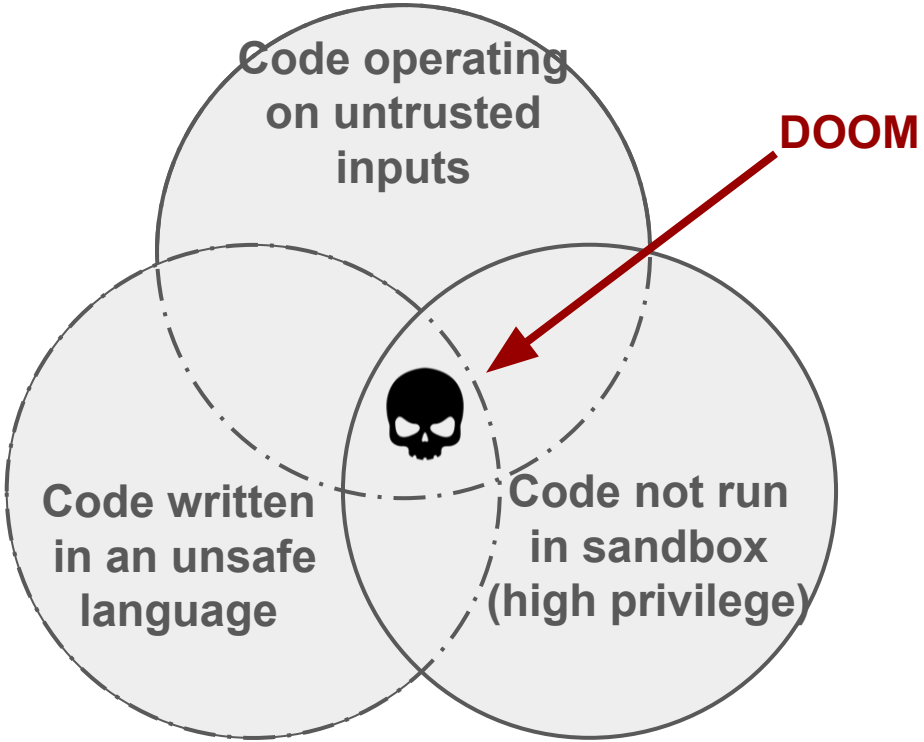
Can we make existing C/C++ code memory safe?

- Google: 90% of Android vulnerabilities [illegible] fety issues
- Apple: 60-70% of vulnerabil[illegible] emory safety vulnerabilities
- Alex Gayn[illegible] bilities were memory sa[illegible]

RETROFITTING SECURITY WAS NOT SUFFICIENT

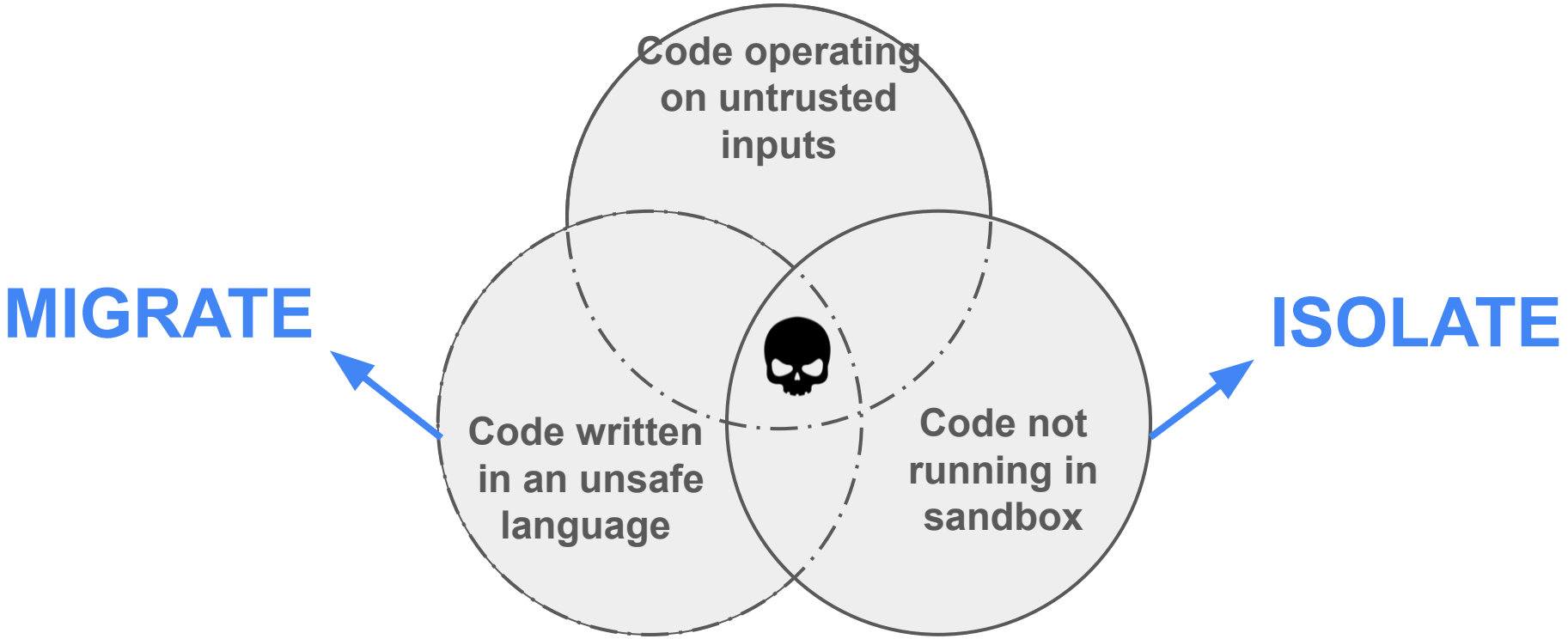
https://security.googleblog.com/2019/05/queue-hardening-enhancements.htm
https://langui.sh/2019/07/23/apple-memory-safety/
https://twitter.com/LazyFishBarrel/status/1129000965741404160

Chromium’s Rule of Two



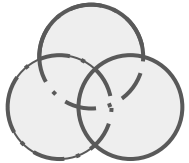
https://chromium.googlesource.com/chromium/src/+/master/docs/security/rule-of-2.md

Chromium's Rule of Two



https://chromium.googlesource.com/chromium/src/+/master/docs/security/rule-of-2.md

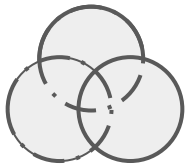
When to migrate and when to isolate?

MIGRATE ←  → **ISOLATE**

- Buy-in from current developers
- Expertise in source & target language
- Resources available to do the porting
- Resources available to maintain port

- Software isolation overheads tolerable
- Hardware isolation feasible
- Cannot support original and rewritten versions of same component

When to migrate and when to isolate?

MIGRATE ←  → **ISOLATE**

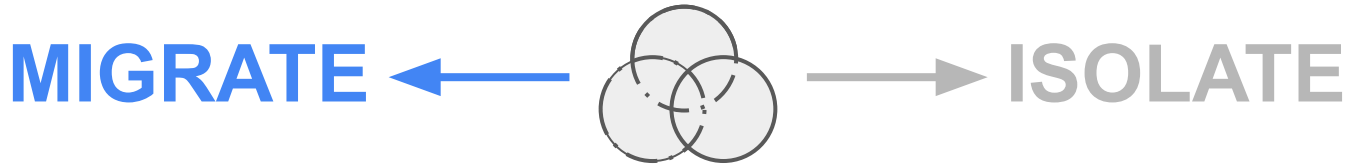
- Buy-in from current developers
- Expertise in source & target language
- Re[illegible]ing
- Re[illegible]maintain port

productivity gains from modern languages

- Software isolation overheads tolerable
- Hardware isolation feasible
- [illegible]itten

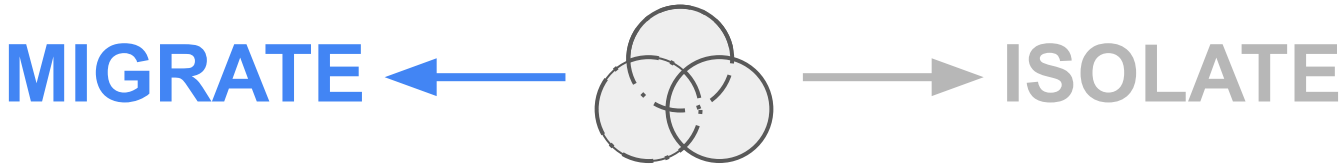
meaningful security benefits in less time

When to migrate and when to isolate?



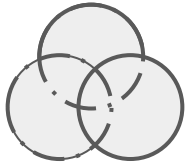
- Decide on a memory safe language
  - Chromium’s strategy
    - Java for Android-specific code
    - Swift for iOS-specific code
    - Rust for third-party use
    - JavaScript or WebAssembly
  - C# good option for Microsoft-specific code
  - Swift for Apple-specific code

When to migrate and when to isolate?



Step 1: Decide what to migrate

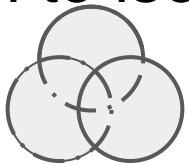
When to migrate and when to isolate?

MIGRATE ←  → **ISOLATE**

Step 1: Decide what to migrate

https://www.usenix.org/system/files/sec22-alexopoulos.pdf

When to migrate and when to isolate?

MIGRATE ←  → **ISOLATE**

How Long Do Vulnerabilities Live in the Code? A Large-Scale Empirical Measurement Study on FOSS Vulnerability Lifetimes

Nikolaos Alexopoulos, Manuel Brack, Jan Philipp Wagner, Tim Grube and Max Mühlhäuser
Telecooperation Lab, Technical University of Darmstadt, Germany

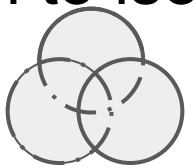
Abstract

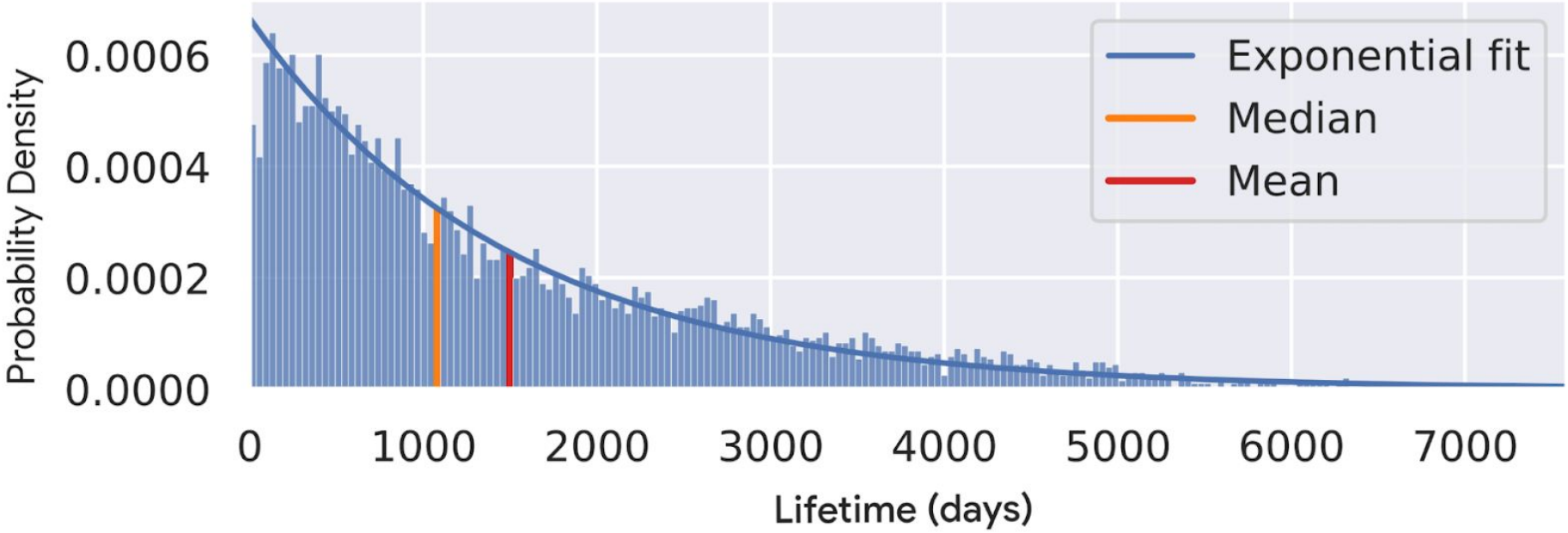
How long do vulnerabilities live in the repositories of large, evolving projects? Although the question has been identified as an interesting problem by the software community in online forums, it has not been investigated yet in adequate depth and scale, since the process of identifying the exact point in time when a vulnerability was introduced is particularly cumbersome. In this paper, we provide an automatic approach for accurately estimating how long vulnerabilities remain in the code (their *lifetimes*). Our method relies on the observation that while it is difficult to pinpoint the exact point of

Reducing the number of vulnerabilities in software by finding existing ones and avoiding the introduction of new ones (e.g. by employing secure coding practices or formal verification techniques) is one of the primary pursuits of computer security.

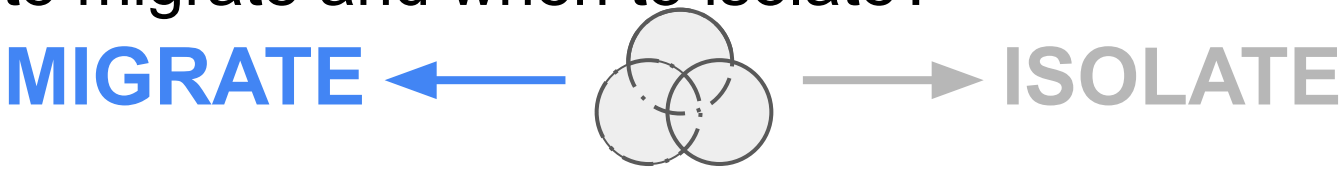
Measurement studies on the different stages of the *vulnerability lifecycle* play an important role in this pursuit, as they help us better understand the impact of security efforts and improve software security practices and workflows. The community has produced a number of such outputs in recent years [5, 8, 14, 18, 23, 28, 36]. A vulnerability's lifecycle, or

When to migrate and when to isolate?

MIGRATE ←  → **ISOLATE**



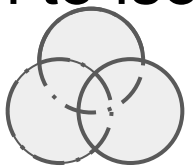
When to migrate and when to isolate?

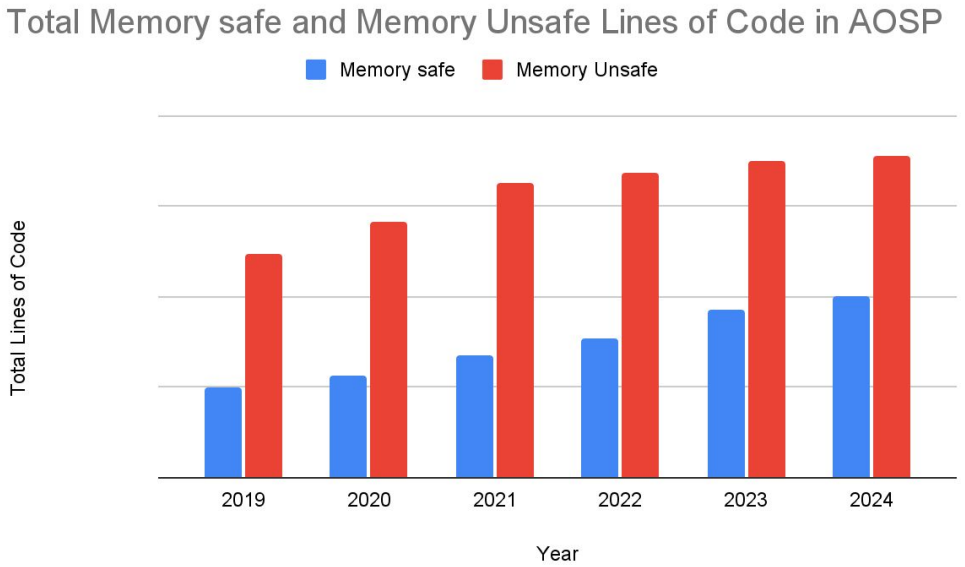


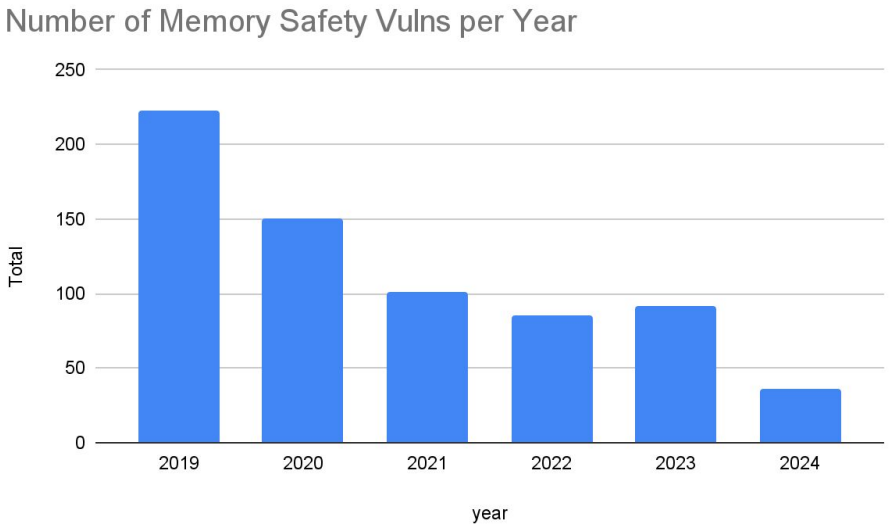
1. **Most memory safety problems in new code**
2. **Code gets safer over time, exponentially**

https://security.googleblog.com/2024/09/eliminating-memory-safety-vulnerabilities-Android.html

When to migrate and when to isolate?

MIGRATE ←  → **ISOLATE**

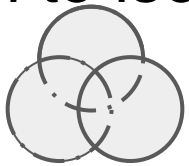




https://security.googleblog.com/2024/09/eliminating-memory-safety-vulnerabilities-Android.html

When to migrate and when to isolate?

MIGRATE ←

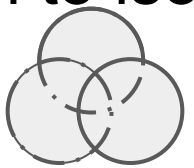


Gradual migration → language interoperability

- Different memory management models
- Different type systems
- Different exception handling mechanisms
- Different data structure layouts and representations
- Different meta-programming facilities
- Different build systems
- …

When to migrate and when to isolate?

MIGRATE ←



Gradual migration → language interoperability

- Interoperability with C is doable
  - Simple type system ↔ complex type system
  - bindgen and cbindgen for Rust
- Interoperability with C++ is hard
  - Complex type system ↔ complex type system
  - Cxx, autocxx, Crubit for Rust

How to migrate a particular module/feature/library?

MANUALLY

- Best when existing code is poorly structured or lacks sufficient tests
- Not all features need to be ported

UNSAFE TRANSLATION
AUTOMATICALLY

- Best when existing code is high quality and tests have high code coverage
- Drop-in compatibility desired
- Need to keep up with upstream

How to migr ary?

MAN CALLY

- Best when exis code is high quality structured or la code coverage
- Not all features y desired th upstream

PROSSIMO

Support this Work

The First Stable Release of a Memory Safe sudo Implementation

Josh Aas
Aug 29, 2023

Prossimo is pleased to announce the first stable release of sudo-rs, our Rust rewrite of the critical sudo utility.

The sudo utility is one of the most common ways for engineers to cross the privacy boundary between user and administrative accounts in the ubiquitous Linux operating system. As such, its security is of the utmost importance.

The sudo-rs project improves on the security of the original sudo by:

- Using a memory safe language (Rust), as it's estimated that one out of three security bugs in the original sudo have been memory management issues
- Leaving out less commonly used features so as to reduce attack surface
- Developing an extensive test suite which even managed to find bugs in the original sudo

How to migr ary?

MAN CALLY

- Best when exis code is high quality structured or la code coverage
- Not all features y desired th upstream

PROSSIMO

Support this Work

Porting C to Rust for a Fast and Safe AV1 Media Decoder

Stephen Crane
Sep 9, 2024

> AV1 is an increasingly important video format and it needs a memory safe, high performance decoder. We worked with the team at Immunant to develop **rav1d**, a Rust-based port of **dav1d**, a C decoder. This is the first of two blog posts about how the team approached this effort.
>
> — Josh Aas, Head of ISRG's Prossimo project

Complex data parsing is one of the most security-critical operations in modern software. Browsers must decode untrusted audio and video inputs encoded with extremely complicated formats in real time. Memory safety bugs in this decoding process are disastrous and common. For example, researchers fuzzing H.264 decoder implementations have demonstrated that these decoders are a dangerous source of bugs. AV1 is a similarly complex, widely used video format. We need a memory safe, performant implementation of AV1 format parsing to avoid parsing vulnerabilities in heavily targeted software such as browsers.

To create this fast and safe AV1 decoder, we have ported an existing high performance AV1 decoding library, **dav1d**, from C to Rust: **rav1d**. Our implementation is drop-in compatible with the **dav1d** C API. Format

Quick aside on Internet Security Research Group

ISRG

Funders

Google

aws

Fly.io

FUTUREWEI Technologies

CISCO

acton family giving

Sovereign Tech Fund

Alpha-Omega

craig newmark philanthropies

Chainguard

CLOUDFLARE

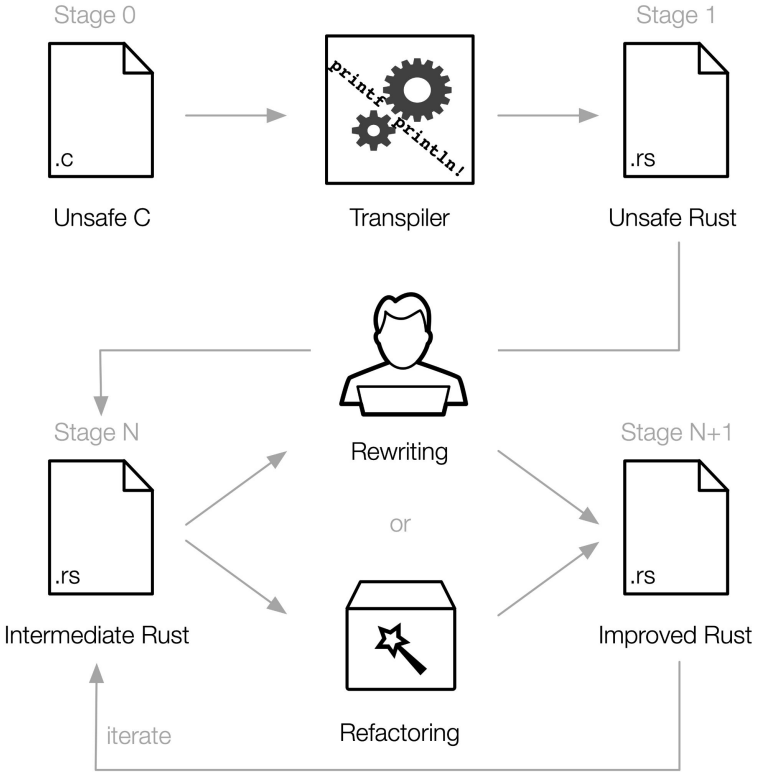
shopify

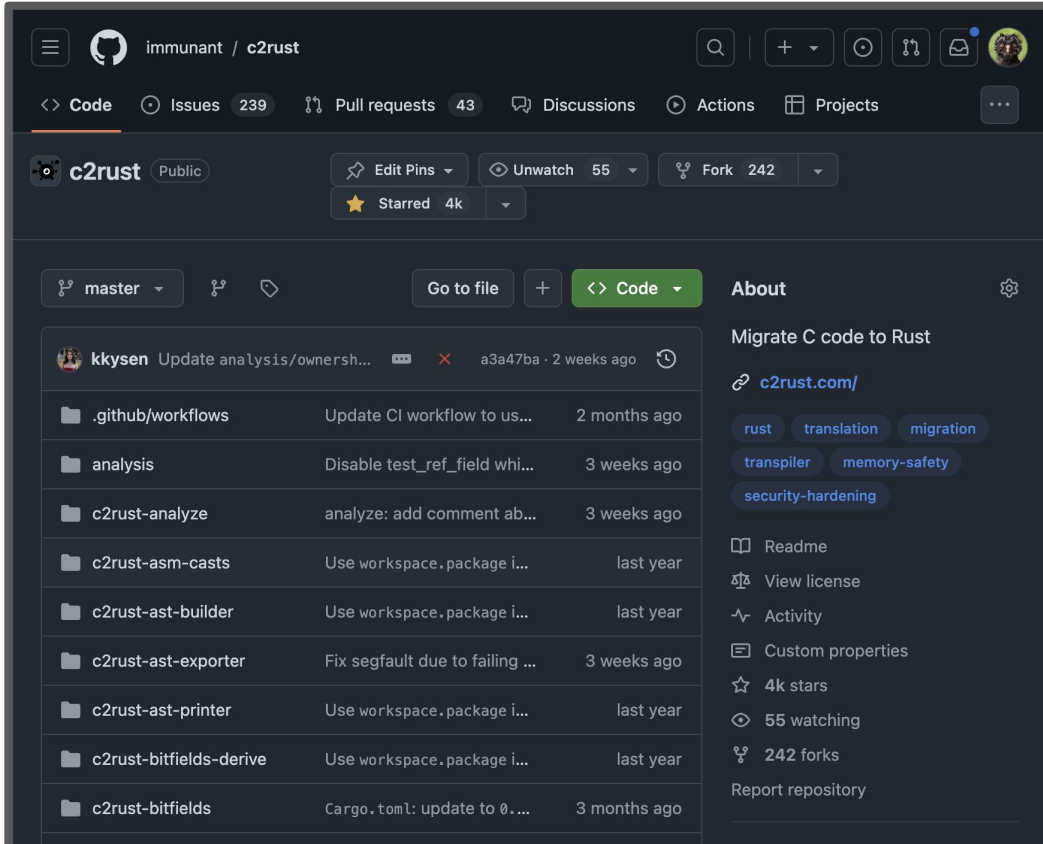
Josh Aas (Co-Founder)

Sarah Gran (VP of Comms)

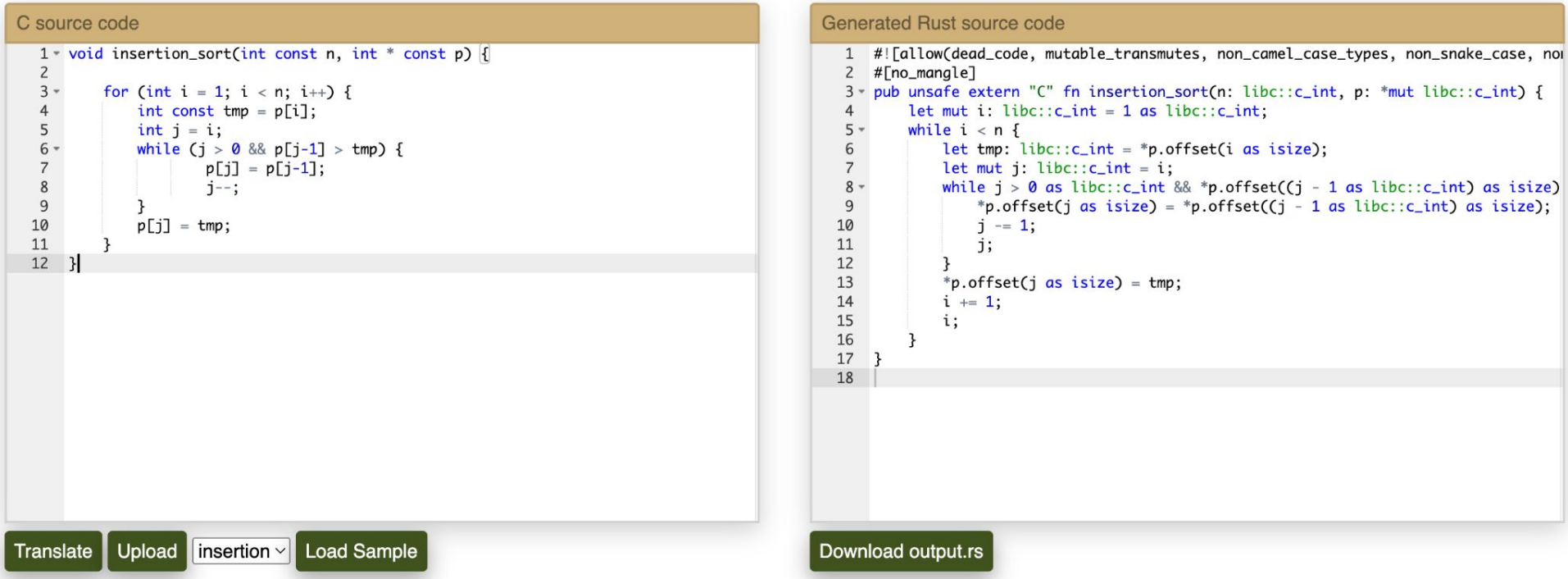
https://www.memorysafety.org/ and https://isrg.org/

Porting dav1d from C to Rust using c2rust





C2Rust transpiler example inputs and outputs



Try it out at https://c2rust.com or https://github.com/immunant/c2rust

C2Rust transpiler example inputs and outputs

FAR FROM THE DESIRED OUTPUT

Try it out at https://c2rust.com or https://github.com/immunant/c2rust

Porting dav1d from C to Rust using c2rust

- Initial transpile took almost no time (~1 day)
  - Transpilation was beneficial w.r.t. testing & tracking upstream changes
  - Transpiler output was ~5x as verbose as input C code
- ~90% time spent “cleaning up” the transpiler output (~12mos)
  - Full test suite run on each commit to catch regressions early
  - Extensive use of C preprocessor macros presented a challenge
  - Threading patterns difficult to translate to Rust efficiently
- ~10% time spent optimizing, handling language differences, CI config, etc.
  - Difficult to track down where we lost performance vs. the C version
    - Currently seeing a ~6% overhead on x86_64
  - Details https://www.memorysafety.org/blog/rav1d-performance-optimization/

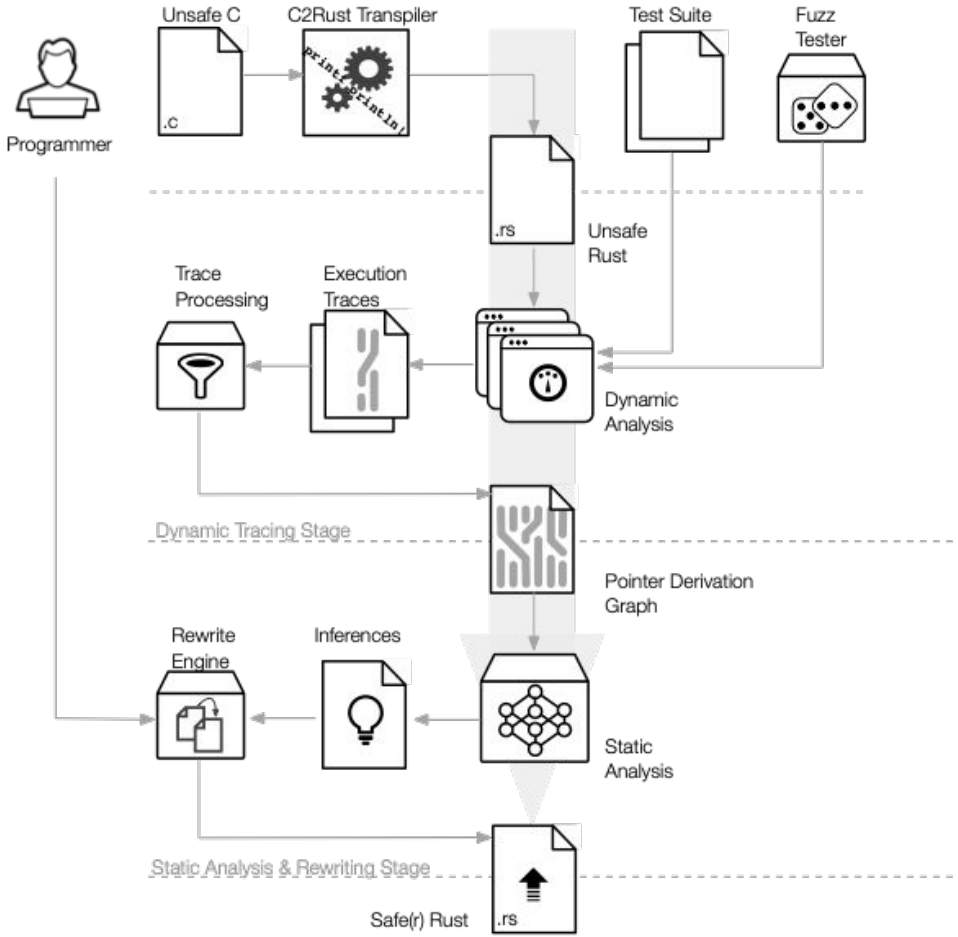
Porting dav1d from C to Rust using c2rust

- Initial transpile took almost no time (~1 day)
  - Transpilation was beneficial w.r.t. testing
- ~90% time spent "cleaning" (unsafe)
  - Full test suite run
  - Extensive
  - Thread
- ~10% time spent on build scripts, CI config, etc.
  - Difficult to
    - Cur

WE NEED MORE AUTOMATION TO LOWER COSTS

Lifting C code to Safe Rust

- Use static analysis to identify lifting opportunities
- Use dynamic analysis to augment static analysis
- Combine results in to pointer derivation graph
  - Assign permissions to pointers
- Rewriting engine maps C types to safe Rust types



immunant x |galois|

Lifting C code to Safe Rust

- Use static analysis to identify lifting opportunities
- Use dynamic analysis to augment static analysis
- Combine results in to pointer derivation graph
  - Assign permissions to pointers
- Rewriting engine maps C types to safe Rust types

Write	Unique	Free	Offset	Ptr type
				`&T`
✓	✓			`&mut T`
✓				`Cell<T>`
			✓	`&[T]`
✓	✓		✓	`&mut [T]`
✓/✗	✓	✓		`Box`
✓/✗	✓	✓	✓	`Box<[T]>`

Heap Allocations

```rust
pub fn buffer_init() -> DynOwned<Box<buffer>> {
    // was: calloc(1, size_of::<buffer>())
    let b: DynOwned<Box<buffer>> =
        DynOwned::new(Box::new(buffer {
            ptr: None,
            used: 0,
            size: 0,
        }));
    return b.take();
}
```

```rust
pub struct buffer {
    // was: *mut i8
    pub ptr: Option<DynOwned<Box<[i8]>>>,
    pub used: u32,
    pub size: u32,
}
```

(DynOwned<T> tracks ownership dynamically, like Option<T>)

Heap Allocations

```
pub fn buffer_init() -> DynOwned<Box<buffer>> {
    // was: calloc(1, size_of::<buffer>())
    let b: DynOwned<Box<buffer>> =
        DynOwned::new(Box::new(buffe
            ptr: None,
            used: 0
            size
        }));
    return b.take(
}
```

```
pub                ffer {
                   ut i8
    ptr: Option<DynOwned<Box<[i8]>>>,
pub used: u32,
pub size:
```

SCALING PROGRAM ANALYSIS AND REFACTORING IS HARD

(DynOwned<T> tr ship dynamically, like Option<T>)

DARPA TRACTOR

https://www.darpa.mil/program/translating-all-c-to-rust

Current state of the art (not all translations are equally good)

Original C code

```c
int main(int argc, char **argv) {
    printf("Hello, %s\n", argv[1]);
}
```

Safe, idiomatic translation (hand-written)

```rust
fn main() {
    let name = std::env::args().nth(1)
      .unwrap_or(&String::from(""));

    println!("Hello, {}", name);
}
```

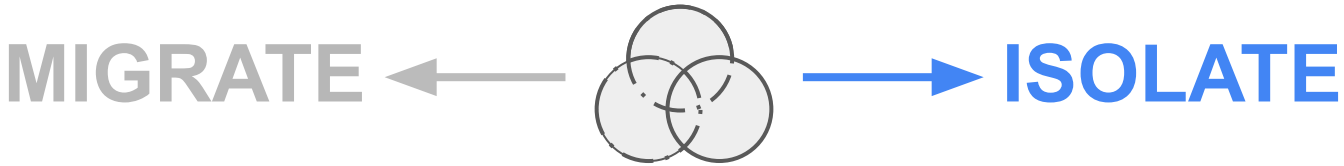
Bug-for-bug compatible (will crash if args are missing)

Supplies empty string if missing arguments

Unsafe Rust translation (via c2rust)

```rust
#![allow(dead_code, mutable_transmutes, non_camel_case_types, non_snake_case,
non_upper_case_globals, unused_assignments, unused_mut)]
#![register_tool(c2rust)]
#![feature(register_tool)]
extern "C" {
    fn printf(_: *const libc::c_char, _: ...) -> libc::c_int;
}
unsafe fn main_0(
    mut argc: libc::c_int,
    mut argv: *mut *mut libc::c_char,
) -> libc::c_int {
    printf(
        b"Hello, %s\n\0" as *const u8 as *const libc::c_char,
        *argv.offset(1 as libc::c_int as isize),
    );
    return 0;
}
pub fn main() {
    let mut args: Vec::<*mut libc::c_char> = Vec::new();
    for arg in ::std::env::args() {
        args.push(
            (::std::ffi::CString::new(arg))
                .expect("Failed to convert argument into CString.")
                .into_raw(),
        );
    }
    args.push(::std::ptr::null_mut());
    unsafe {
        ::std::process::exit(
            main_0(
                (args.len() - 1) as libc::c_int,
                args.as_mut_ptr() as *mut *mut libc::c_char,
            ) as i32,
        )
    }
}
```

When to migrate and when to isolate?



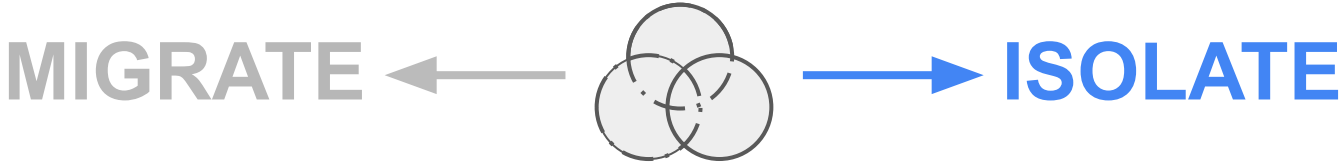
- Hardware-based isolation
  - CHERI
  - TrustZone
  - Intel SGX
  - AMD SVM
  - (Apple Secure Enclave)

When to migrate and when to isolate?

- Hardware-based isolation
  - CHERI
  - TrustZ
  - Intel S
  - AMD S
  - (Apple

NOT PORTABLE ACROSS DIFFERENT PROCESSORS

When to migrate and when to isolate?



- Virtualization
  - Android Virtualization Framework
    - https://source.android.com/docs/core/virtualization/architecture
  - Extends Linux KVM with protected VM feature
  - Supports both ARM and x86_64

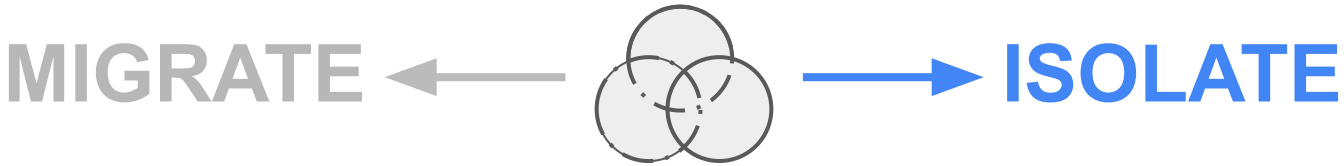
When to migrate and when to isolate?

- Virtualization
  - Android Virtuali-
    - ht
  - Exter
  - Suppo

zation/architecture

FLEXIBLE BUT ALSO HEAVYWEIGHT AND HIGHLY COMPLEX

When to migrate and when to isolate?



- Process level-sandboxing
  - Separate high and low privilege components in separate OS processes
  - Used by most modern browsers
  - Highly portable and mature isolation strategy
    - Non-trivial to apply to monolithic applications

When t

- Proc
  - S
  - U
  - H

ses

ENIGMA

https://www.usenix.org/conference/enigma2021/presentation/palmer

THE LIMITS OF SANDBOXING AND NEXT STEPS

Note: Presentation times are in Pacific Standard Time (PST).

Wednesday, February 03, 2021 - 9:50 am–10:20 am
Chris Palmer, Google Chrome Security

Abstract:
Privilege separation and reduction ("sandboxing") has significantly improved software security, and in many applications is a baseline requirement for safe design. (In fact, there are still many applications that can and should adopt sandboxing.)

Although necessary, sandboxing is not sufficient by itself. The designs and implementations of real-world operating systems put a ceiling on the effectiveness and applicability of sandboxing. From years of experience shipping Chromium, we have learned that (1) Chromium is at or near the limit of how much safety it can practically provide with privilege separation and reduction; and (2) we still need to provide greater resilience.

Therefore, we must find and develop additional security mechanisms. Our primary approach is now working toward increased memory safety. Where sandboxing limits the value attackers gain from exploiting vulnerabilities, memory-safe(r) code can eliminate vulnerabilities altogether or make it infeasible to use them in an exploit chain.

ENIGMA

When t

https://www.usenix.org/conference/enigma2021/presentation/palmer

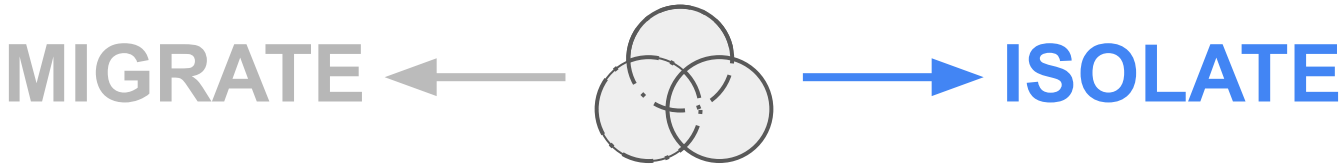
applicability of sandboxing. From years of experience shipping Chromium, we have learned that (1) Chromium is at or near the limit of how much safety it can practically provide with privilege separation and reduction; and (2) we still need to provide greater resilience.

Therefore, we must find and develop additional security mechanisms. Our primary approach is now working toward increased memory safety. Where sandboxing limits the value attackers gain from exploiting vulnerabilities, memory-safe(r) code can eliminate vulnerabilities altogether or make it infeasible to use them in an exploit chain.

shipping Chromium, we have learned that (1) Chromium is at or near the limit of how much safety it can practically provide with privilege separation and reduction; and (2) we still need to provide greater resilience.

Therefore, we must find and develop additional security mechanisms. Our primary approach is now working toward increased memory safety. Where sandboxing limits the value attackers gain from exploiting vulnerabilities, memory-safe(r) code can eliminate vulnerabilities altogether or make it infeasible to use them in an exploit chain.

When to migrate and when to isolate?



- Library-level sandboxing
  - Avoids the need to split application into multiple processes
- WebAssembly
- RLBox
  - https://rlbox.dev/
  - Can use WebAssembly or PKRU as backends
- Hardware-assisted library sandboxing
  - https://github.com/immunant/IA2-Phase2/

Takeaway

- Memory safe languages delivers assurance that previous practices lacks.
- **Follow the rule of two**!
  (don’t process untrusted input with unsafe code)
- We need to dramatically **lower the cost of migrating code** to memory safe languages.


THANK YOU
GRACIAS
ARIGATO
SHUKURIA
BOLZÏN
MERCI
BÏYAN
DANKSCHEEN
TASHAKKUR ATU
SUKSAMA
EKHMET
MEHRBANI
GRAZIE
JUSPAXAR
EFCHARISTO
GOZAIMASHITA
KOMAPSUMNIDA
MAAKE
PALDIES
SHUKRIA
TINGKI
YAQHANYELAY
SPASSIBO
SNACHALHUYA
NUHUN
CHALTU
WABEEJA
MAITEKA
YUSPAGABÄTAM
HUI
GUI
HATUR
UNALCHÉESH
DHANYABAAD
ANIHA
ATTO
MERSI
SPASIBO
DENKAUJA
NENACHALHYA
EKOJU
SIKOMO
MAKETAU
MINMONCHAR
SANCO
MERASTAWHY
GAEJTHO
LAH
TAVTAPUCH
MEDAWAGSE
AGUYJE
FAKAAUE
BARKA

Memory Safety

- Means that software or a programming language is designed to catch memory bugs
- Memory safe languages automate memory management only allow safe memory accesses
- Most languages are memory safe
- C/C++ are the only languages that are still widely used but not memory safe
  - Prizes efficiency and flexibility, programmer is in full control and has full responsibility
  - Routinely used to write low-level, highly-privileged code
- Consequences of Memory Unsafety
  - Security vulnerabilities
  - Program instability
  - Difficult debugging and maintenance
- Further reading
  - https://www.internetsociety.org/resources/doc/2023/how-to-talk-to-your-manager-about-memory-safety
  - https://www.memorysafety.org/docs/memory-safety/
  - https://storage.googleapis.com/gweb-research2023-media/pubtools/7665.pdf

Further Reading

- https://www.internetsociety.org/resources/doc/2023/how-to-talk-to-your-manager-about-memory-safety
- https://www.memorysafety.org/docs/memory-safety/
- https://storage.googleapis.com/gweb-research2023-media/pubtools/7665.pdf