Credits & Acknowledgments

Rust

...for being a breath of fresh air

Kernel maintainers

...for being open-minded

Everyone that has helped Rust for Linux

(see credits in the RFC & patch series)
History

30 years of Linux

30 years of ISO C
Love story*

30 years of Linux  ❤  30 years of ISO C

* Terms and Conditions Apply.
An easy task?
An easy task?

“Do you see any language except C which is suitable for development of operating systems?”
An easy task?

“Do you see any language except C which is suitable for development of operating systems?”

“I like interacting with hardware from a software perspective. And I have yet to see a language that comes even close to C.”

— Linus Torvalds 2012
Why is C a good language for the kernel?

“You can use C to generate good code for hardware.”

“When I read C, I know what the assembly language will look like.”

“The people that designed C ... designed it at a time when compilers had to be simple.”

“If you think like a computer, writing C actually makes sense.”

Fast
Low-level
Simple
Fits the domain
But...
But...
So, what does Rust offer?
So, what does Rust offer?
Safety

Safety in Rust

= No undefined behavior
Safety

Safety in Rust

≠

Safety in “safety-critical”

as in functional safety (DO-178B/C, ISO 26262, EN 50128…)
Is avoiding UB that important?
Is avoiding UB that important?

~70% of vulnerabilities in C/C++ projects come from UB

See more at https://www.memorysafety.org/docs/memory-safety/
Sure, UB is an issue and safe Rust does not have it…
Sure, UB is an issue and safe Rust does not have it…

...does Rust really help, though?
Does Rust help?

Derived using data from https://adalogics.com/blog/fuzzing-100-open-source-projects-with-oss-fuzz
What else does Rust offer?

Language
What else does Rust offer?

- Stricter type system
- Safe/unsafe split
- Modules & visibility
- Generics
- Lifetimes
- Pattern matching
- Shared & exclusive references
- RAlII
- Sum types
- Powerful hygienic and procedural macros
What else does Rust offer?

Freestanding standard library
What else does Rust offer?

- Freestanding standard library
- Vocabulary types like `Result` and `Option`
- Pinning
- Formatting
- Checked, saturating & wrapping integer arithmetic primitives
- Iterators
What else does Rust offer?

Tooling
What else does Rust offer?

- Tooling
  - Great compiler error messages
  - Unit & integration tests
  - Static analyzer
  - Linter
  - Formatter
  - IDE tooling
  - Macro debugging
  - C ↔ Rust bindings generators
  - UBSAN-like interpreter
  - Great compiler error messages
What else does Rust offer?

- Documentation generator
- Unit & integration tests
- Static analyzer
- C ↔ Rust bindings generators
- Linter
- Macro debugging
- Formatter
- IDE tooling
- Great compiler error messages
- UBSAN-like interpreter

*plus the usual friends: gdb, lldb, perf, valgrind*
What is the catch?
What is the catch?

Cannot model everything \Rightarrow Unsafe code required
What is the catch?

Cannot model everything => Unsafe code required

More information to provide => More complex language
What is the catch?

Cannot model everything \[\Rightarrow\] Unsafe code required

More information to provide \[\Rightarrow\] More complex language

Extra runtime checks \[\Rightarrow\] Potentially expensive
What is the catch?

- Cannot model everything ⇒ Unsafe code required
- More information to provide ⇒ More complex language
- Extra runtime checks ⇒ Potentially expensive
- An extra language to learn ⇒ Logistics & maintenance burden
Why is C a good language for the kernel?

“You can use C to generate good code for hardware.”

“When I read C, I know what the assembly language will look like.”

“The people that designed C ... designed it at a time when compilers had to be simple.”

“If you think like a computer, writing C actually makes sense.”
Why is C a good language for the kernel?

“You can use C to generate good code for hardware.”

“When I read C, I know what the assembly language will look like.”

“The people that designed C ... designed it at a time when compilers had to be simple.”

“If you think like a computer, writing C actually makes sense.”

Fast: Yes

Low-level: Sometimes

Simple: Not really

Fits the domain ...
An easy task?

“Do you see any language except C which is suitable for development of operating systems?”

“I like interacting with hardware from a software perspective. And I have yet to see a language that comes even close to C.”

— Linus Torvalds 2012
An easy task? *maybe?*

“Do you see any language except C which is suitable for development of operating systems?”

“I like interacting with hardware from a software perspective. And I have yet to see a language that comes even close to C.”

— Linus Torvalds 2012
Some examples where Rust helps
```rust
pub fn main() {
    let a = Box::new(42);
    drop(a);
    drop(a);
}
```

```
Compilation failed.

# For more information see the output

# To open the output window, click or
```

Error [E0382]: use of moved value: `a`

```rust
let a = Box::new(42);
    move occurs because `a` has type `Box<isize>`, which does not implement the `Copy` trait
    value moved here
    value used here after move
```

Error: aborting due to previous error.

For more information about this error, try `rustc --explain E0382`.

```c
#include <stdlib.h>

int main(void)
{
    int * const a = malloc(sizeof(int));
    if (a == NULL)
        abort();
    *a = 42;
    free(a);
    free(a);
}
```

Program returned: 139
Program stderr:
free(): double free detected in tcache 2
```c
#include <stdlib.h>

// On success, 'result' is written and the return value is 0.
// On error, the return value is < 0.
int get_some_data(int *result);
void do_something(int foo);

void f(void) {
    int data;
    if (get_some_data(&data) < 0)
        abort();
    do_something(data);
}
```

```rust
extern "C"
fn get_some_data() -> Result<132, &>;
fn do_something(Bar: 132);

pub fn f() {
    let data = get_some_data().unwrap();
    do_something(data);
}
```
Building an abstraction
// Bindings
extern {
    fn get_pointer() -> *mut i32;
    fn use_pointer(ptr: *mut i32);
}
extern {
    fn get_pointer() -> *mut i32;
    fn use_pointer(ptr: *mut i32);
}

mod foo {
    /// # Invariants
    /// The pointer is valid.
    pub struct Foo {
        ptr: *mut i32,
    }

    impl Foo {
        pub fn new() -> Self {
            Foo {
                ptr: unsafe { crate::get_pointer() }
            }
        }

        pub fn do_something(&mut self) {
            unsafe { crate::use_pointer(self.ptr); }
        }
    }
}
// Bindings
extern {
    fn get_pointer() -> *mut i32;
    fn use_pointer(ptr: *mut i32);
}

// Abstractions code
mod foo {
    /// Invariants
    /// The pointer is valid.
    pub struct Foo {
        ptr: *mut i32,
    }

    impl Foo {
        pub fn new() -> Self {
            Foo {
                // SAFETY: `get_pointer()` is always safe to call.
                ptr: unsafe { crate::get_pointer() }
            }
        }

        pub fn do_something(&mut self) {
            // SAFETY: `use_pointer()` requires that the pointer
            // is valid, which holds due to the type invariant.
            unsafe { crate::use_pointer(self.ptr); }
        }
    }

    // User code
    use foo::*;

    pub fn f() {
        let mut my_foo = Foo::new();
        my_foo.do_something();
    }
}
// Bindings
extern {
    fn get_pointer() -> *mut i32;
    fn use_pointer(ptr: *mut i32);
}

// Abstractions code
mod foo {
    /// Invariants
    ///
    /// The pointer is valid.
    pub struct Foo {
        ptr: *mut i32,
    }

    impl Foo {
        pub fn new() -> Self {
            Foo {
                // SAFETY: `get_pointer()` is always safe to call.
                ptr: unsafe { crate::get_pointer() }
            }
        }

        pub fn do_something(&mut self) {
            // SAFETY: `use_pointer()` requires that the pointer
            // is valid, which holds due to the type invariant.
            unsafe { crate::use_pointer(self.ptr); }
        }
    }
}

// User code
use foo::*;

pub fn f() {
    let mut my_Foo = Foo::new();
    my_Foo.ptr = 42 as *mut i32;
    my_Foo.do_something();
}
Rust support in the kernel
Driver point of view
Forbidden!

Safe

Unsafe

Abstractions

my_foo driver

foo/

bindgen

include/

bindings crate

foo subsystem

bar subsystem

kernel crate

drivers/
Supported architectures

- arm (armv6 only)
- arm64
- powerpc (ppc64le only)
- riscv (riscv64 only)
- x86 (x86_64 only)

See Documentation/rust/arch-support.rst
Supported architectures

arm (armv6 only)
arm64
powerpc (ppc64le only)
riscv (riscv64 only)
x86 (x86_64 only)

See Documentation/rust/arch-support.rst

...so far!
32-bit and other restrictions should be easy to remove
Kernel LLVM builds work for mips and s390
GCC codegen paths should open up more
Rust codegen paths for the kernel

- **rustc_codegen_gcc**: Already passes most rustc tests
- **rustc_codegen_llvm**: Main one
- **Rust GCC**: Expected in 1-2 years (rough estimate)
Documentation
The kernel crate.

This crate contains the kernel APIs that have been ported or wrapped for usage by Rust code in the kernel and is shared by all of them.

In other words, all the rest of the Rust code in the kernel (e.g. kernel modules written in Rust) depends on `core`, `alloc` and this crate.

If you need a kernel C API that is not ported or wrapped yet here, then do so first instead of bypassing this crate.

**Modules**

- **buffer**: Struct for writing to a pre-allocated buffer with the `write!` macro.
- **c_types**: C types for the bindings.
- **chrdev**: Character devices.
- **file**: Files and file descriptors.
- **file_operations**: File operations.
- **io_buffer**: Buffers used in IO.
- **iov_iter**: IO vector iterators.
- **linked_list**: Linked lists.
- **miscdev**: Miscellaneous devices.
- **of**: Devicetree and Open Firmware abstractions.
- **pages**: Kernel page allocation and management.
- **platdev**: Platform devices.
- **prelude**: The kernel prelude.
- **print**: Printing facilities.
Struct kernel::sync::Mutex

```rust
pub struct Mutex<T: ?Sized> { /* fields omitted */ }
```

- Exposes the kernel's `struct mutex`. When multiple threads attempt to lock the same mutex, only one at a time is allowed to progress, the others will block (sleep) until the mutex is unlocked, at which point another thread will be allowed to wake up and make progress.

A `Mutex` must first be initialised with a call to `Mutex::init` before it can be used. The `mutex_init` macro is provided to automatically assign a new lock class to a mutex instance.

Since it may block, `Mutex` needs to be used with care in atomic contexts.

### Implementations

```rust
impl<T> Mutex<T>
```

- pub unsafe fn new(t: T) -> Self

  Constructs a new mutex.

### Safety

The caller must call `Mutex::init` before using the mutex.

```rust
impl<T: ?Sized> Mutex<T>
```

-
/// A string that is guaranteed to have exactly one `NUL` byte, which is at the
/// end.
///
/// Used for interoperability with kernel APIs that take C strings.
#[repr(transparent)]
pub struct CStr([u8]);

impl CStr {
    /// Returns the length of this string excluding `NUL`.
    #[inline]
    pub const fn len(&self) -> usize {
        self.len_with_nul() - 1
    }

    /// Returns the length of this string with `NUL`.
    #[inline]
    pub const fn len_with_nul(&self) -> usize {
        // SAFETY: This is one of the invariants of `CStr`.
        // We add a `unreachable_unchecked` here to hint the optimizer that
        // the value returned from this function is non-zero.
        if self.0.is_empty() {
            unsafe { core::hint::unreachable_unchecked(); }
        }
        self.0.len()
    }
}
### Results for `pr`

<table>
<thead>
<tr>
<th>In Names (176)</th>
<th>In Parameters (0)</th>
<th>In Return Types (0)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>kernel::print</code></td>
<td>Printing facilities.</td>
<td></td>
</tr>
<tr>
<td><code>kernel::platdev::PlatformDriver::probe</code></td>
<td>Platform driver probe.</td>
<td></td>
</tr>
<tr>
<td><code>kernel::pr_err</code></td>
<td>Prints an error-level message (level 3).</td>
<td></td>
</tr>
<tr>
<td><code>kernel::pr_cont</code></td>
<td>Continues a previous log message in the same line.</td>
<td></td>
</tr>
<tr>
<td><code>kernel::pr_crit</code></td>
<td>Prints a critical-level message (level 2).</td>
<td></td>
</tr>
<tr>
<td><code>kernel::pr_info</code></td>
<td>Prints an info-level message (level 6).</td>
<td></td>
</tr>
<tr>
<td><code>kernel::pr_warn</code></td>
<td>Prints a warning-level message (level 4).</td>
<td></td>
</tr>
<tr>
<td><code>kernel::prelude</code></td>
<td>The kernel prelude.</td>
<td></td>
</tr>
<tr>
<td><code>kernel::pr_alert</code></td>
<td>Prints an alert-level message (level 1).</td>
<td></td>
</tr>
<tr>
<td><code>kernel::pr_emerg</code></td>
<td>Prints an emergency-level message (level 0).</td>
<td></td>
</tr>
<tr>
<td><code>kernel::linked_list::CursorMut::peek_prev</code></td>
<td>Returns the element immediately before the one the cursor ...</td>
<td></td>
</tr>
<tr>
<td><code>kernel::pr_notice</code></td>
<td>Prints a notice-level message (level 5).</td>
<td></td>
</tr>
<tr>
<td><code>kernel::prelude::Vec::swap_remove</code></td>
<td>Removes an element from the vector and returns it.</td>
<td></td>
</tr>
<tr>
<td><code>kernel::prelude::Box::is_prefix_of</code></td>
<td>The alloc Prelude</td>
<td></td>
</tr>
<tr>
<td><code>kernel::prelude::Box::strip_prefix_of</code></td>
<td>The libcore prelude</td>
<td></td>
</tr>
<tr>
<td><code>alloc::prelude</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>core::prelude</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>core::iter::Product</code></td>
<td>Trait to represent types that can be created by ...</td>
<td></td>
</tr>
<tr>
<td><code>core::iter::Product::product</code></td>
<td>Method which takes an iterator and generates <code>Self</code> from ...</td>
<td></td>
</tr>
<tr>
<td><code>core::iter::iterator::product</code></td>
<td>Iterates over the entire iterator, multiplying all the ...</td>
<td></td>
</tr>
<tr>
<td><code>core::option::Option::product</code></td>
<td>Takes each element in the <code>[Iterator]; if it is a </code>[None]`, ...</td>
<td></td>
</tr>
</tbody>
</table>
Documentation code
/// Wraps the kernel's `struct task_struct`.
///
/// # Invariants
///
/// The pointer `Task::ptr` is non-null and valid. Its reference count is also non-zero.
///
/// # Examples
///
/// The following is an example of getting the PID of the current thread with
/// zero additional cost when compared to the C version:
///
/// ```
/// #
/// use kernel::prelude::*;
/// use kernel::task::Task;
///
/// # fn test() {
/// Task::current().pid();
/// # } 
/// #
///
/// pub struct Task {
///   pub(crate) ptr: *mut bindings::task_struct,
/// }
Conditional compilation

Rust code has access to conditional compilation based on the kernel config

#[cfg(CONFIG_X)] // `CONFIG_X` is enabled (`y` or `m`)
#[cfg(CONFIG_X="y")] // `CONFIG_X` is enabled as a built-in (`y`)
#[cfg(CONFIG_X="m")] // `CONFIG_X` is enabled as a module (`m`)
#[cfg(not(CONFIG_X))] // `CONFIG_X` is disabled
Coding guidelines

- No direct access to C bindings
- No undocumented public APIs
- No implicit `unsafe` block
- Docs follows Rust standard library style
- `// SAFETY` proofs for all unsafe blocks
- Clippy linting enabled
- Automatic formatting enforced

Rust 2018 edition & idioms
- No unneeded panics
- No infallible allocations

...
Coding guidelines

- No direct access to C bindings
- No undocumented public APIs
- No implicit unsafe block
- Docs follows Rust standard library style
  // SAFETY proofs for all unsafe blocks
- Clippy linting enabled
- Automatic formatting enforced

Rust 2018 edition & idioms
- No unneeded panics
- No infallible allocations

Aiming to be as strict as possible
Abstractions code
/** Wraps the kernel's `struct file`.
**/

/// # Invariants
/// The pointer `File::ptr` is non-null and valid.
/// Its reference count is also non-zero.

pub struct File {
    pub(crate) ptr: *mut bindings::file,
}
impl File {
    /// Constructs a new [\`struct file\`] wrapper from a file descriptor.
    ///
    /// The file descriptor belongs to the current process.
    pub fn from_fd(fd: u32) -> Result<Self> {
        // SAFETY: FFI call, there are no requirements on `fd`.
        let ptr = unsafe { bindings::fget(fd) };
        if ptr.is_null() {
            return Err(Error::EBADF);
        }

        // INVARIANTS: We checked that `ptr` is non-null, so it is valid.
        // `fget` increments the ref count before returning.
        Ok(Self { ptr })
    }

    // ...
}
Driver code
static int pl061_resume(struct device *dev)
{
    int offset;

    struct pl061 *pl061 = dev_get_drvdata(dev);

    for (offset = 0; offset < PL061_GPIO_NR; offset++) {
        if (pl061->csave_regs.gpio_dir & (BIT(offset)))
            pl061_direction_output(&pl061->gc, offset,
                    pl061->csave_regs.gpio_data & (BIT(offset)));
        else
            pl061_direction_input(&pl061->gc, offset);
    }

    writeb(pl061->csave_regs.gpio_is, pl061->base + GPIOIS);
    writeb(pl061->csave_regs.gpio_ibe, pl061->base + GPIOIBE);
    writeb(pl061->csave_regs.gpio_iev, pl061->base + GPIOIEV);
    writeb(pl061->csave_regs.gpio_ie, pl061->base + GPIOIE);

    return 0;
}

fn resume(data: &Ref<DeviceData>) -> Result {
    let inner = data.lock();
    let pl061 = data.resources().ok_or(Error::ENXIO)?;

    for offset in 0..PL061_GPIO_NR {
        if inner.csave_regs.gpio_dir & bit(offset) != 0 {
            let v = inner.csave_regs.gpio_data & bit(offset) != 0;
            let _ = <Self as gpio::Chip>::direction_output(  
                    data, offset.into(), v);
        } else {
            let _ = <Self as gpio::Chip>::direction_input(  
                    data, offset.into());
        }
    }

    pl061.base.writeb(inner.csave_regs.gpio_is, GPIOIS);
    pl061.base.writeb(inner.csave_regs.gpio_ibe, GPIOIBE);
    pl061.base.writeb(inner.csave_regs.gpio_iev, GPIOIEV);
    pl061.base.writeb(inner.csave_regs.gpio_ie, GPIOIE);

    Ok(())}
Testing code
fn trim_whitespace(mut data: &[u8]) -> &[u8] {
    // ...
}

#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn test_trim_whitespace() {
        assert_eq!(trim_whitespace(b"foo    "), b"foo");
        assert_eq!(trim_whitespace(b"    foo"), b"foo");
        assert_eq!(trim_whitespace(b"  foo  "), b"foo");
    }
}
/// Getting the current task and storing it in some struct. The reference count is automatically
/// incremented when creating `State` and decremented when it is dropped:
///
/// # use kernel::prelude::*;
/// use kernel::task::Task;
///
/// struct State {
///     creator: Task,
///     index: u32,
/// }
///
/// impl State {
///     fn new() -> Self {
///         Self {
///             creator: Task::current().clone(),
///             index: 0,
///         }
///     }
/// }
///
/// ```
How to proceed?
https://github.com/Rust-for-Linux/linux
Rust for Linux

Miguel Ojeda
ojeda@kernel.org
Backup slides
C Charter

6. **Keep the spirit of C.** The Committee kept as a major goal to preserve the traditional spirit of C. There are many facets of the spirit of C, but the essence is a community sentiment of the underlying principles upon which the C language is based. The C11 revision added a new facet to the original list of facets. The new spirit of C can be summarized in phrases like:

(a) Trust the programmer.
(b) Don't prevent the programmer from doing what needs to be done.
(c) Keep the language small and simple.
(d) Provide only one way to do an operation.
(e) Make it fast, even if it is not guaranteed to be portable.
(f) Make support for safety and security demonstrable.

— N2086 C2x Charter - Original Principles

12. **Trust the programmer, as a goal, is outdated in respect to the security and safety programming communities.** While it should not be totally disregarded as a facet of the spirit of C, the C11 version of the C Standard should take into account that programmers need the ability to check their work.

— N2086 C2x Charter - Additional Principles for C11
Undefined Behavior

3.4.3 undefined behavior

behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this document imposes no requirements

2 Note 1 to entry: Possible undefined behavior ranges from ignoring the situation completely with unpredictable results, to behaving during translation or program execution in a documented manner characteristic of the environment (with or without the issuance of a diagnostic message), to terminating a translation or execution (with the issuance of a diagnostic message).

3 Note 2 to entry: J.2 gives an overview over properties of C programs that lead to undefined behavior.

4 EXAMPLE An example of undefined behavior is the behavior on dereferencing a null pointer.
Example of UB

— The value of the second operand of the / or % operator is zero (6.5.5).

```c
int f(int a, int b) {
    return a / b;
}
```
Example of UB

— The value of the second operand of the / or % operator is zero (6.5.5).

```c
int f(int a, int b) {
    return a / b;
}
```

UB \( \forall x \ f(x, 0) \);
Example of UB

Any other inputs that trigger UB?

```c
int f(int a, int b) {
    return a / b;
}
```
Example of UB

Any other inputs that trigger UB?

```c
int f(int a, int b) {
    return a / b;
}
```

UB `f(INT_MIN, -1);`
Instances of UB
Instances of UB

— The value of the second operand of the / or % operator is zero (6.5.5).
Instances of UB

The execution of a program contains a data race (5.1.2.4).

The second operand of the / or % operator is zero (6.5.5).
Instances of UB

- Execution of a program contains a data race (5.1.2.4).
- The second operand of the / or % operator is zero (6.5.5).
- An object is referred to outside of its lifetime (6.2.4).
The value of a pointer to an object whose lifetime has ended is used (6.2.9).

An object is referred to outside of its lifetime (6.2.4).

The second operand of the operator is zero (6.5.5).

A second operand of the operator contains a data race (5.1.2.4).

Instances of UB
An object is referred to outside of its lifetime (6.2.4).

The value of an object with automatic storage duration is indeterminate (6.2.4).

the value of a pointer.

The value of a pointer.
Instances of `UB`

- The value of a pointer whose lifetime has ended is used while it is indeterminate (6.2.4, 6.7.9, 6.8).
- A trap representation is read by an lvalue expression that does not have character type (6.2.6.1).
- Execution of a program whose lifetime has ended is used (6.2.4).
- The second operand of the `operator` is zero (6.5.5).
- An object is referred to outside of its lifetime (6.2.4).
Instances of UB

- An object is "outside of its lifetime" (6.2.4).
- Pointers that do not point into, or just beyond, the same array object are subtracted (6.2.4).
- A trap representation is read by an lvalue expression that does not have character type (6.2.4).
- The value of an object with duration is indeterminate (6.2.4).
- "A second operand of the operator whose lifetime is used while it is indeterminate (6.5.6)."
- "A data race (5.1.2.4)."
Instances of UB

- The value of a pointer to outside of its lifetime (6.2.4).

- A trap representation is read by an lvalue expression that does not have character type (6.2.6.1).

- The value of an object whose lifetime is not fully specified (6.7.9, 6.8).

- Execution of a program whose lifetime or the result of an operator is not specified (6.5.6).

- The second operand of the * operator is out of its lifetime (6.2.4).

- Pointers that do not point into, or just beyond, the same array object are subtracted (6.2.4).
int f(int a, int b) {
    if (b == 0)
        abort();

    if (a == INT_MIN && b == -1)
        abort();

    return a / b;
}
Avoiding UB

```c
int f(int a, int b) {
    if (b == 0)
        abort();

    if (a == INT_MIN && b == -1)
        abort();

    return a / b;
}
```

*f is a safe function*
Safe function

```c
int f(int a, int b) [[safe]] { 
    if (b == 0) 
        abort();

    if (a == INT_MIN && b == -1) 
        abort();

    return a / b;
}
```

f is a safe function
Safe function

```c
int f(int a, int b) [[safe]] {
    if (b == 0)
        abort();
    if (a == INT_MIN && b == -1)
        abort();
    return a / b;
}
```

f is a safe function

Not C (yet? N2659)
Safety examples

\[ \Rightarrow \]

`abort()`s in C are Rust-safe
Safety examples

abort()s in C are Rust-safe

⇒

Even if your company goes bankrupt.
Safety examples

abort()s in C

are

Rust-safe

⇒

Even if your company goes bankrupt.

Even if somebody is injured.
Safety examples

⇒

Rust panics are Rust-safe
Safety examples

⇒

Kernel panics are Rust-safe
Safety examples

Uses after free, null derefs, double frees, OOB accesses, uninitialized memory reads, invalid inhabitants, data races...

⇒

are not

Rust-safe
Safety examples

Uses after free, null derefs, double frees, OOB accesses, uninitialized memory reads, invalid inhabitants, data races...

⇒

are not Rust-safe

Even if your system still works.
Safety examples

⇒

Race conditions

are

Rust-safe
Safety examples

⇒

Memory leaks are Rust-safe
Safety examples

⇒

Deadlocks are Rust-safe
Safety examples

⇒

Integer overflows are Rust-safe
Is avoiding UB that important?

~70% of the vulnerabilities Microsoft assigns a CVE each year continue to be memory safety issues

— https://msrc-blog.microsoft.com/2019/07/18/we-need-a-safer-systems-programming-language/
Is avoiding UB that important?

Mojave (aka macOS 10.14)

Apple released macOS 10.14 Mojave on September 24, 2018 and subsequently has issued 6 point releases.

<table>
<thead>
<tr>
<th>Total CVE Count</th>
<th>Memory Unsafety Bugs</th>
<th>Percentage</th>
<th>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>36</td>
<td>81.8%</td>
<td>10.14.6</td>
</tr>
<tr>
<td>45</td>
<td>40</td>
<td>88.9%</td>
<td>10.14.5</td>
</tr>
<tr>
<td>38</td>
<td>20</td>
<td>52.6%</td>
<td>10.14.4</td>
</tr>
<tr>
<td>23</td>
<td>22</td>
<td>95.7%</td>
<td>10.14.3</td>
</tr>
<tr>
<td>13</td>
<td>11</td>
<td>84.6%</td>
<td>10.14.2</td>
</tr>
<tr>
<td>71</td>
<td>40</td>
<td>56.3%</td>
<td>10.14.1</td>
</tr>
<tr>
<td>64</td>
<td>44</td>
<td>68.8%</td>
<td>10.14</td>
</tr>
</tbody>
</table>

Is avoiding UB that important?

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.

Is avoiding UB that important?

Most of Android’s vulnerabilities occur in the media and bluetooth components. Use-after-free (UAF), integer overflows, and out of bounds (OOB) reads/writes comprise 90% of vulnerabilities with OOB being the most common.

— https://security.googleblog.com/2019/05/queue-hardening-enhancements.html
Is avoiding UB that important?

Fish in a Barrel @LazyFishBarrel · Sep 9
5/8 vulnerabilities fixed in Firefox 92 are memory unsafety mozilla.org/en-US/security... #memoryunsafety

Security Vulnerabilities fixed in Firefox 92
mozilla.org

Fish in a Barrel @LazyFishBarrel · Sep 1
13/19 (5/5 high) vulnerabilities fixed in Google Chrome 93.0.4577.63 are memory unsafety chromereleases.googleblog.com/2021/08/stable... #memoryunsafety

Stable Channel Update for Desktop
The Chrome team is delighted to announce the promotion of Chrome 93 to the stable channel for ... chromereleases.googleblog.com
Does Rust help?
Does Rust help?

I took a look at this spreadsheet published a couple weeks ago...
Does Rust help?

I took a look at this spreadsheet published a couple weeks ago...

Fuzzing 100+ open source projects with OSS-Fuzz - lessons learned.

31st August, 2021

David Korczynski & Adam Korczynski, Security Research & Security Engineering

---

**Project specs**

<table>
<thead>
<tr>
<th>Project name</th>
<th>Github URL</th>
<th>Language</th>
<th>Bugs</th>
<th>Security Bugs</th>
<th>Bugs verified (fixed)</th>
<th>Security bugs verified (fixed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>apache-ht叩d</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>blacktide</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>caddy</td>
<td>8</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>cascadia</td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cctz</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>ctena</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>cllium</td>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Cletaweb</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Clib</td>
<td>11</td>
<td>0</td>
<td></td>
<td></td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>containerd</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dgraph</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Does Rust help?

I filled the language column and plotted...
Does Rust help?

I filled the language column and plotted...