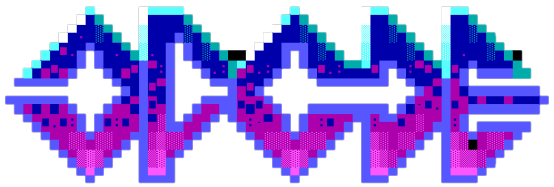
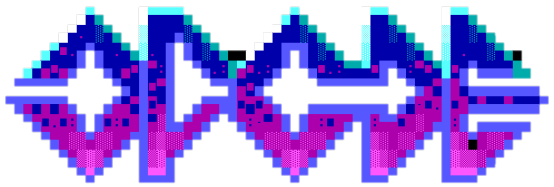
How unsafe {} is Rust?

Matt Suiche, OPCDE (@OPCDE)



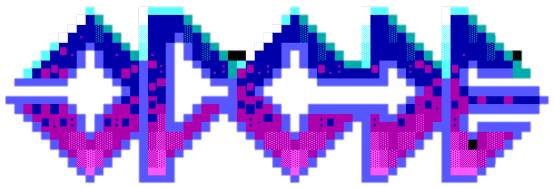
Whoami

- @msuiche on Twitter
- Founder of Hackito Ergo Sum (Paris), No Such Con (Paris), OPCDE (Dubai & Nairobi) conferences
- CloudVolumes (Vmware in 2014), Comae (Magnet Forensics in 2022).
- Memory acquisition & analysis research
- Windows ARM64 exploit development research
- Web3 security



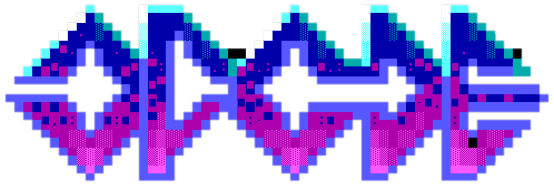
Rust vs Go

- Rust is memory safe, and ensures memory safety at the time of compilation.
- Go is not memory safe and sometimes data races can lead to invalid values which may lead to memory corruption.
- Although, both Go and Rust prevent memory leaks by design. Go relies on its automated garbage collector.
  - Rust ownership and borrowing are pretty awesome too.
  - Rust allows **multiple immutable** references but only a **single changeable reference** at a time.



Rust vs Carbon

- Carbon is an **experimental** successor to C++
- Safer fundamentals, and an **incremental path towards a memory-safe** subset. 🚩



Rust

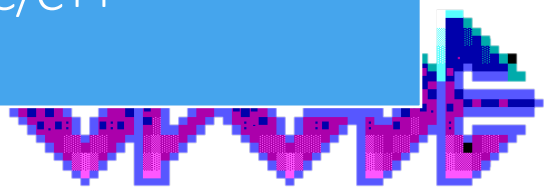
Most beloved programming language for 6 years in a row

Stack Overflow Developer Survey

One language for all?

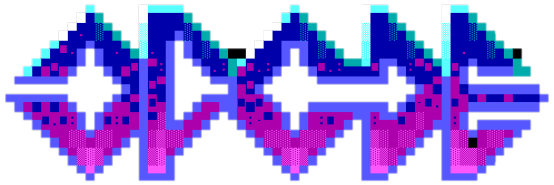
user-mode applications	kernel drivers	Firmwares	Web Applications	Web3 smart contracts (Solana, NEAR, etc.)

Many projects also consider Rust as a viable long-term alternative to C/C++



What type of bugs? (list non-exhaustive)

DoS	Memory issues	Logic bugs	Supply chain
• panic!()	• unsafe{}	• can lead to profit in the case of smart-contracts	• rogue crates/github repo

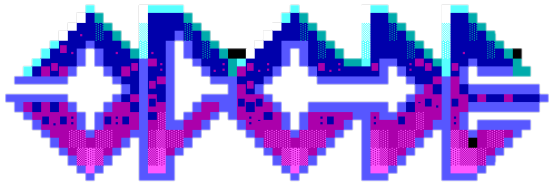


Smart-contract pitfalls (Solana)

5 Categories (Neodyme)

- Missing ownership check
- Missing signer check
- Solana account confusions
- Arbitrary signed program invocation
- Integer overflow & underflow

Example: Soteria

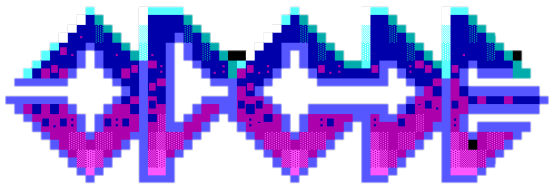


24 programs/jet/src/state/reserve.rs

```
@@ -195,16 +195,16 @@ impl Reserve {
195 195     pub fn deposit(&mut self, token_amount: u64, note_amount: u64) {
196 196         let state = self.state_mut().get_stale_mut();
197 197
198     -         state.total_deposits += token_amount;
199     -         state.total_deposit_notes += note_amount;
    198 +         state.total_deposits = state.total_deposits.checked_add(token_amount).unwrap();
    199 +         state.total_deposit_notes = state.total_deposit_notes.checked_add(note_amount).unwrap();
200 200     }
201 201
202 202     /// Record an amount of tokens withdrawn from the reserve
203 203     pub fn withdraw(&mut self, token_amount: u64, note_amount: u64) {
204 204         let state = self.state_mut().get_stale_mut();
205 205
206     -         state.total_deposits -= token_amount;
207     -         state.total_deposit_notes -= note_amount;
    206 +         state.total_deposits = state.total_deposits.checked_sub(token_amount).unwrap();
    207 +         state.total_deposit_notes = state.total_deposit_notes.checked_sub(note_amount).unwrap();
208 208     }
```

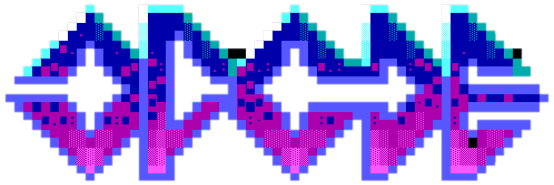
Jet Protocol (October 2021)

Integer overflow/underflow



Unsafety (As per documentation)

- Unsafe operations are those that can potentially violate the memory-safety following guarantees of Rust's static semantics.
- The language level features cannot be used in the safe subset of Rust:
  - Dereferencing a raw pointer.
  - Reading or writing a mutable or external static variable.
  - Accessing a field of a union, other than to assign to it.
  - Calling an unsafe function (including an intrinsic or foreign function).
  - Implementing an unsafe trait.

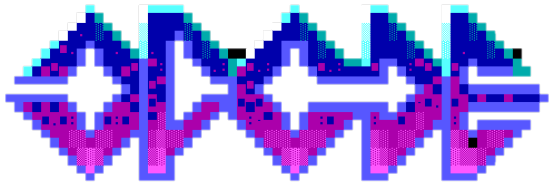



Petr Beneš
@PetrBenes
official rust wdk when
1:46 AM · Oct 31, 2022 · Twitter Web App


Steve Eckels
@stevemk14ebr
Rust isn't ready! What needs to change:
* MS makes native rust winapi bindings. For kernel too!
* MS supports DDK + rustc together
* Rust adds anonymous unions and structures
* Disabling unwinding w/ no_std needs to be easier (i.e. actually work)
* Stack usage limit flags!!
Mark Russinovich @markrussinovich · Sep 20
Speaking of languages, it's time to halt starting any new projects in C/C++ and use Rust for those scenarios where a non-GC language is required. For the sake of security and reliability. the industry should declare those languages as deprecated.
1:54 AM · Sep 21, 2022 · Twitter for Android

Supply Chain Attacks

- Non-Official Rust WDK
  - High risk of potential supply chain abuse
  - Main branch: https://github.com/retep998/winapi-rs
  - ARM64: https://github.com/msuiche/winapi-rs
- Reserved crate names
  - https://users.rust-lang.org/t/should-people-be-allowed-to-reserve-crate-names/8360
- CrateDepression
  - SentinelLabs has investigated a supply-chain attack against the Rust development community that we refer to as ‘CrateDepression’.
  - https://www.sentinelone.com/labs/cratedepression-rust-supply-chain-attack-infects-cloud-ci-pipelines-with-go-malware/

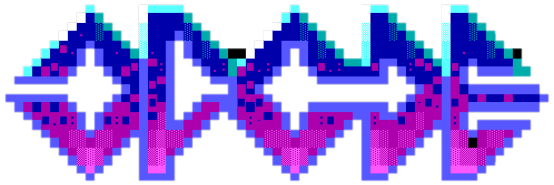


Memory Safety (e.g. CVE-2019-18960)

- CVE-2019-18960 in Aws Firecracker. Amazon VM monitor.
  - Write up by Valentina Palmiotti, @chompie1337
- Guard page at the end of signal stacks (#69969)

```
1  pub fn get_host_address(&self, guest_addr: GuestAddress) -> Result<*const u8> {
2          self.do_in_region(guest_addr, 1, |mapping, offset| {
3              // This is safe; `do_in_region` already checks that offset is in
4              // bounds.
5              Ok(unsafe { mapping.as_ptr().add(offset) } as *const u8)
6          })
       }
```

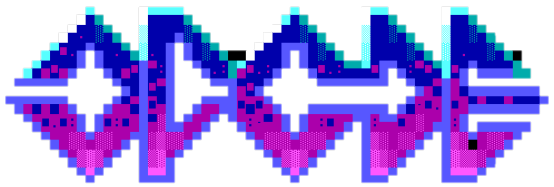
A memory region base address and the `offset` of the guest address from the base is calculated in `do_in_region` and the addition of the two is returned as the resulting pointer. On line 5 in the code snippet above, there is an `unsafe` block. In Rust, a block of code can be prefixed with the `unsafe` keyword to permit operations such as dereferencing a raw pointer, reading or writing to a mutable static variable, accessing a field of a union (other than to assign it), or calling an `unsafe` function **[10]**. In the code snippet above, the comment states that the operation in the `unsafe` block is safe to allow because `do_in_region` checks that the offset is in bounds. Let's take a look:



unsafe user-mode code (raw pointers)

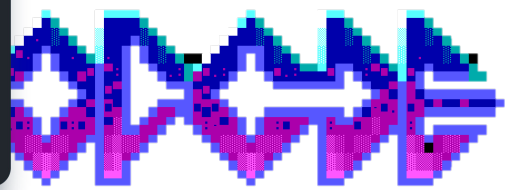
- “`as_ptr()`” and “`as *const`” references.
  - Raw, unsafe pointers, *const T, and *mut T.
  - Direct old school pointer manipulation.

```
// Create an image path string value
let image_path = CString::new("ImagePath").unwrap();
if RegSetValueExA(key_handle,
                  image_path.as_ptr(),
                  0,
                  REG_SZ,
                  nt_driver_path.as_ptr() as *const u8,
                  path_length) != 0 {
        return Some(GetLastError());
}
```



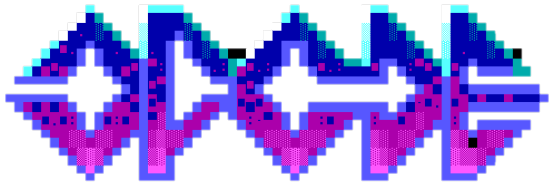
unsafe kernel-mode code

```rust
pub fn get_physical_memory_ranges() -> Result<Vec<PhysicalMemoryRange>, NTSTATUS> {
    unsafe {
        let mut buf = MmGetPhysicalMemoryRanges();

        if buf.is_null() {
            return Err(STATUS_ACCESS_DENIED);
        }

        let mut ranges = Vec::new();

        loop {
            if (*buf).number_of_bytes == 0 {
                break;
            }

            ranges.push(*buf);
            buf = buf.wrapping_add(1);
        }

        Ok(ranges)
    }
}
```



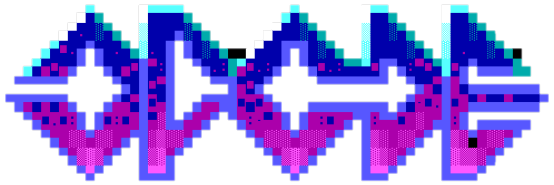
Interesting projects

- Linux will support the Rust programming language in its kernel from version 6.1.
- Caliptra
  - An open-source root of trust with firmware written in Rust.
  - https://azure.microsoft.com/en-us/blog/delivering-consistency-and-transparency-for-cloud-hardware-security/
  - An open sourced register-transfer level (RTL) code implementation of Caliptra that can be synthesized into current SoC designs will be made available, along with the cloud-designed firmware written entirely in Rust.



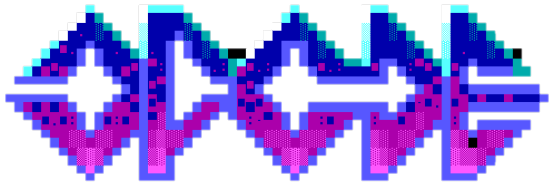
Stay safe

- https://rustsec.org
  - Tools
  - Advisory Database
- Soteria
- Tools
  - cargo audit
  - cargo deny
  - cargo geiger
  - cargo fuzz
  - miri
  - Other tools leveraging llvm (KLEE, rust-san, etc.)



Acknowledgments

- Patrick Ventuzelo (@Pat_Ventuzelo)
- Neodyme
- Matt Miller (@epakskape)
- Taha Karim (@lordx64)
- Jonas Lyk (@jonasLyk)
- Ryan McCrystal
- Not Mathias
- Stephan van Schaik
- Valentina Palmiotti (@chompie1337)
- Memn0ps (@memN0ps)
- Petr Benes (@PetrBenes)
- Steve Eckels (@stevemk14ebr)



Questions?