Enhancing Security in Polkadot and Substrate: Navigating Rust and Blockchain Vulnerabilities

By Mohammadreza Ashouri | PhD in Cyber Security

Sub0 Asia 2024 - Bangkok

ashourics@protonmail.com

About me

- PhD in Cyber Security 2020 (Potsdam)
- Postdoc in System Security 2021 (VT)
- Blog: https://ashourics.medium.com
- YouTube: https://www.youtube.com/c/Heapzip
- Github: https://github.com/mohammadreza-ashouri/
- Linkedin: https://www.linkedin.com/in/drashouri/

Feel free to contact me via Email (ashourics@protonmail.com)
or Telegram: @bytescan

What is Rust?

- safer alternative to C and C++
- system programming language
- statically typed language
- enhance memory safety and error detection

Rust Security Model

- Ownership and type system
- Security model is enforced by its borrow checker
- Borrow checker -> your Rust code is memory-safe and has no data races
- Like garbage collector in Java but cooler!

Let’s check the borrow checker!

- Is there anything wrong with this C code?

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
    char *x = strdup("Hello");
    free(x);
    printf("%s\n", x);
}
```

Let’s check the borrow checker!

- What happens in Rust?

```rust
fn main() {
    let x = String::from("Hello");
    drop(x);
    println!("{}", x);
}
```

```
error[E0382]: borrow of moved value: `x`
 --> src/main.rs:4:20
  |
2 |     let x = String::from("Hello");
  |         - move occurs because `x` has type `String`, which does not implement the `Copy` trait
3 |     drop(x);
  |          - value moved here
4 |     println!("{}", x);
  |                    ^ value borrowed here after move
  |
  = note: this error originates in the macro `$crate::format_args_nl` which comes from the expansion o
help: consider cloning the value if the performance cost is acceptable
  |
3 |     drop(x.clone());
  |           ++++++++
```

Rust Unsafe!

- What happens in unsafe Rust?

```rust
fn main() {
    let x = String::from("Hello");
    let y: *const String = &x; // Create a raw pointer to `x`.
    drop(x); // Drop `x`, which deallocates the memory `x` was managing.
    unsafe {
        println!("{}", *y); // Dereference the raw pointer. This is undefined behavior!
    }
}
```

```
Finished dev [unoptimized + debuginfo] target(s) in 1.26s
```

https://www.acsac.org/2020/program/poster-wips/2020-3-RUSTY%20%20A%20Fuzzing%20Tool%20for%20Rust.pdf

Limitations of the Rust Security Model?

- Memory leaks
- Complexity

https://wiki.sei.cmu.edu/confluence/display/c/MEM31-C.+Free+dynamically+allocated+memory+when+no+longer+needed

Complexity

```
fn main() {
    let mut v = vec![1, 2, 5];
    let mut it = v.iter();
    assert_eq!(*it.next().unwrap(), 1);
    v[2] = 3;
    assert_eq!(*it.next().unwrap(), 2);
}
```

Complexity

```rust
fn main() {
    let mut v = vec![1, 2, 5];
    let mut it = v.iter();
    assert_eq!(*it.next().unwrap(), 1);
    v[2] = 3;
    assert_eq!(*it.next().unwrap(), 2);
}
```

```
error[E0502]: cannot borrow `v` as mutable because it is also borrowed as immutable
 --> src/main.rs:5:5
  |
3 |     let mut it = v.iter();
  |                  - immutable borrow occurs here
4 |     assert_eq!(*it.next().unwrap(), 1);
5 |     v[2] = 3;
  |     ^ mutable borrow occurs here
6 |     assert_eq!(*it.next().unwrap(), 2);
  |                 -- immutable borrow later used here

For more information about this error, try `rustc --explain E0502`.
```

Complexity

```
1  #include <cassert>
2  #include <iostream>
3  #include <vector>
4
5  int main() {
6      std::vector<int> v{1,2,5};
7      std::vector<int>::iterator it = v.begin();
8      assert(*it++ == 1);
9      v[2] = 3;   /* memory-safe */
10     assert(*it++ == 2);
11 }
```

Read more: https://doc.rust-lang.org/reference/behavior-considered-undefined.html

Borrow checker is not everything!

- Memory leaks
- Logical Issues
- Lack of data verification & Sanitization
- Wrong security policies

- Unit testing
- Fuzz testing
- Manual code review

CLI Tools to enhance your code security and performance

- Cargo-audit -> Cargo.lock , RustSec Advisory Database
- Clippy -> insecure random number generators, etc
- Cargo-geiger - > minimizing the use of unsafe code

What is Substrate?

- Rust-based open-source toolset introduced by Parity

Create blockchains:

- These blockchains can be standalone networks or designed to connect with the Polkadot network a parachains.

Develop Pallets:

- Pallets are modular components that encapsulate specific functionalities within the Substrate framework.
- Developers use Substrate to create new pallets that can be reused across different blockchain proje

Build Smart Contracts:

- While Substrate itself is more focused on the foundational aspects of blockchain development,
- it also supports smart contract development through the use of pallets like the FRAME Contracts pallet.

Common Vulnerabilities

- Insecure Randomness
- Storage Exhaustion
- Unsafe Arithmetic
- Unsafe Conversion
- Replay Issues
- Outdated Crates

Insecure Randomness

- It arises from using weak or predictable randomness sources, which could potentially be manipulated or anticipated by malicious actors.

- Randomness Collective Flip Pallet (Insecure Approach)
- Why? because the randomness is directly influenced by past block hashes, which could potentially be manipulated or anticipated.

- The Randomness Collective Flip pallet utilizes the hashes of the previous 81 blocks to generate a random value.

- Mitigation?
- Use VRF (Verifiable Random Function) from the Pallet BABE.

Storage Exhaustion

- This issue happens when there is no proper charging for using storage.
- Example: when the cost of storage for users is very cheap, here in this case attacker can exploit low storage costs to bloat the storage, making the system sluggish and expensive to maintain.

- Mitigation : You need to implement checks to ensure the cost charged to users is proportional to the storage used, and consider setting limits on the amount of data that can be saved to storage to prevent any exploitations.

Unsafe Arithmetic

- When you compile your code in the debug mode, arithmetic operations will panic (crash) on overflow or underflow.
- But, in the release mode, Rust performs “wrapping” arithmetic, where values wrap around on overflow or underflow. So, in blockchain code, arithmetic is often used to handle critical operations like token transfers, so this wrapping arithmetic can become risky!
- For example in token transfer scenarios, it could lead to incorrect account balances, enabling attackers to gain unauthorized assets.

- Mitigation? using safe math functions like checked_add or checked_sub, which check for arithmetic errors.

Unsafe Conversion

- Converting one numerical type to another without proper checks, potentially causing errors and exploitations.
- exploits here can lead to overflows or incorrect values, and that allows attackers to cause unexpected behavior.
- Mitigation: you need to ensure we you proper checks during type conversion, avoid down-casting, and use safe conversion methods like unique_saturated_into

Replay Issues

- This vulnerability arises from improper handling of transaction nonces. So this can may allow attackers to repeat transactions and perform some sort of DoS attack and slow down the network.

- Mitigation: ensuring nonces are correctly set-up in the system logic and implementing checks to prevent transaction repetition

Outdated Crates

- Outdated, unsafe, or incompatible versions of dependencies (crates), this inconsistency can expose your code to different vulnerabilities and incompatibility issues.

```
[package]
name = "sample-pallet"
description = "Sample pallet with incoherent dependencies"
version = "1.0.0"
edition = "2021"

[dependencies]
codec = { package = "parity-scale-codec", version = "3.6.1" }
scale-info = { version = "2.5.0", features = ["derive"] }
log = { version = "0.4.14", default-features = false }

frame-system = { version = "4.0.0-dev", default-features = false, git =
"https://github.com/paritytech/substrate.git", branch = "polkadot-v1.0.0" }
frame-support = { version = "4.0.0-dev", default-features = false, git =
"https://github.com/paritytech/substrate.git", branch = "polkadot-v0.9.0" }
pallet-balances = { version = "4.0.0-dev", default-features = false, git =
"https://github.com/paritytech/substrate.git", branch = "polkadot-v0.8.0" }
pallet-message-queue = { version = "7.0.0-dev", default-features = false, git =
"https://github.com/paritytech/substrate.git", branch = "polkadot-v1.0.0" }
pallet-uniques = { version = "4.0.0-dev", default-features = false, git =
"https://github.com/paritytech/substrate.git", branch = "polkadot-v0.9.2" }
```

- Mitigation: Using the newest and safest versions of dependencies, ensuring consistent versioning across all crates

Follow me

- Blog: https://ashourics.medium.com
- YouTube: https://www.youtube.com/c/Heapzip
- Github: https://github.com/mohammadreza-ashouri/
- Linkedin: https://www.linkedin.com/in/drashouri/

Feel free to contact me via Email (ashourics@protonmail.com)
or Telegram: @bytescan