VRust

Security Assessment

O2Lab VRust Team

07/11/2024 09:50:46

Contents

Summary

This report has been prepared for O2Lab VRust Team to discover issues and vulnerabilities in the source code of the O2Lab VRust Team project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques. The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	O2Lab VRust Team
Platform	Ethereum
Language	Solana
Crate	level0
GitHub Location	https://github.com/parasol-aser/vrust
sha256	Unknown

Audit Summary

Delivery Date	07/11/2024
Audit Methodology	Static Analysis
Key Components	

Vulnerability Summary

Vulnerability Level	**Total**
Critical	3
Major	0
Medium	0
Minor	0
Informational	0
Discussion	0

Findings

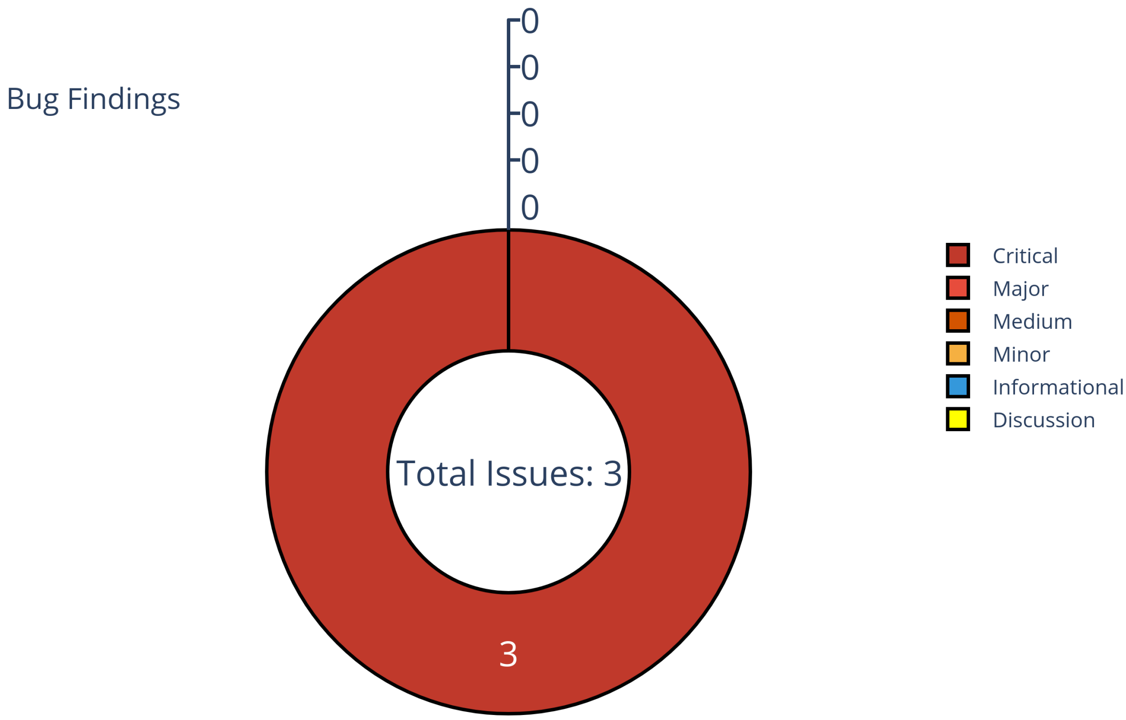

Figure 1: Findings

Finding Statistic

Category	Count
IntegerFlow	1
MissingKeyCheck	2

ID	Category	Severity	Status
0	IntegerFlow	Critical	UnResolved
1	MissingKeyCheck	Critical	UnResolved
2	MissingKeyCheck	Critical	UnResolved

Issue: 0: IntegerFlow

Category	Severity	Status
IntegerFlow	Critical	UnResolved

- Location

level0/src/processor.rs:124:5: 124:49

```
124 **vault_info.lamports.borrow_mut() -= amount
125
```

- Code Context

– Function Definition:

```
102 fn withdraw(_program_id: &Pubkey, accounts: &[AccountInfo], amount: u64) -> ProgramResult
103
```

Vulnerability at Line: 124

```
119     // Vulnerabilità: Invocazione di un programma esterno senza controlli adeguati
120     let cpi_accounts = &[vault_info.clone(), destination_info.clone()];
121     let cpi_instruction = system_instruction::transfer(&vault_info.key, &destination_info.key, amount);
122     invoke(&cpi_instruction, cpi_accounts)?;
123
124     **vault_info.lamports.borrow_mut() -= amount;
125     **destination_info.lamports.borrow_mut() += amount;
126
127     Ok(())
128 }
129
```

- Call Stack

```
fn entrypoint(){// /home/biagio/.cargo/registry/src/github.com-
↪  1ecc6299db9ec823/solana-program-1.8.14/src/entrypoint.rs:120:9: 127:10
↪  }
    fn processor::process_instruction(){// level0/src/processor.rs:15:1:
    ↪  25:2 }
        fn processor::withdraw(){// level0/src/processor.rs:102:1: 128:2 }

```

- description:
- link:
- alleviation:

Issue: 1: MissingKeyCheck

Category	Severity	Status
MissingKeyCheck	Critical	UnResolved

- Location

level0/src/processor.rs:109:46: 109:62

```
109 wallet_info.data
110
```

- Code Context

– Function Definition:

```
102 fn withdraw(_program_id: &Pubkey, accounts: &[AccountInfo], amount: u64) -> ProgramResult
103
```

Vulnerability at Line: 109

```
104     let wallet_info = next_account_info(account_info_iter)?;
105     let vault_info = next_account_info(account_info_iter)?;
106     let authority_info = next_account_info(account_info_iter)?;
107     let destination_info = next_account_info(account_info_iter)?;
108     let cpi_program_info = next_account_info(account_info_iter)?; // Account del programma esterno
109     let wallet = Wallet::deserialize(&mut &(*wallet_info.data).borrow_mut()[..])?;
110
111     assert!(authority_info.is_signer);
112     assert_eq!(wallet.authority, *authority_info.key);
113     assert_eq!(wallet.vault, *vault_info.key);
114
```

- Call Stack

```
fn entrypoint(){// /home/biagio/.cargo/registry/src/github.com-
↪ 1ecc6299db9ec823/solana-program-1.8.14/src/entrypoint.rs:120:9: 127:10
↪ }
    fn processor::process_instruction(){// level0/src/processor.rs:15:1:
    ↪ 25:2 }
        fn processor::withdraw(){// level0/src/processor.rs:102:1: 128:2 }

```

- description:
- link:
- alleviation:

Issue: 2: MissingKeyCheck

Category	Severity	Status
MissingKeyCheck	Critical	UnResolved

- Location

level0/src/processor.rs:90:46: 90:62

```
90 wallet_info.data
91
```

- Code Context

– Function Definition:

```
85 fn deposit(_program_id: &Pubkey, accounts: &[AccountInfo], amount: u64) -> ProgramResult
86
```

Vulnerability at Line: 90

```
85 fn deposit(_program_id: &Pubkey, accounts: &[AccountInfo], amount: u64) -> ProgramResult {
86     let account_info_iter = &mut accounts.iter();
87     let wallet_info = next_account_info(account_info_iter)?;
88     let vault_info = next_account_info(account_info_iter)?;
89     let source_info = next_account_info(account_info_iter)?;
90     let wallet = Wallet::deserialize(&mut &(*wallet_info.data).borrow_mut()[..])?;
91
92     assert_eq!(wallet.vault, *vault_info.key);
93
94     invoke(
95
```

- Call Stack

```
fn entrypoint(){// /home/biagio/.cargo/registry/src/github.com-
↪ 1ecc6299db9ec823/solana-program-1.8.14/src/entrypoint.rs:120:9: 127:10
↪ }
    fn processor::process_instruction(){// level0/src/processor.rs:15:1:
    ↪ 25:2 }
        fn processor::deposit(){// level0/src/processor.rs:85:1: 100:2 }

```

- description:
- link:
- alleviation:

Appendix

Copied from https://leaderboard.certik.io/projects/aave

Finding Categories

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Checksum Calculation Method

The “Checksum” field in the “Audit Scope” section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux “sha256sum” command against the target file.

Disclaimer

Copied from https://leaderboard.certik.io/projects/aave

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.