VRust

Security Assessment

O2Lab VRust Team

28/02/2022 15:29:52

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	solana_escrow
GitHub Location	https://github.com/parasol-aser/vrust
sha256	Unknown

Audit Summary

Delivery Date	28/02/2022
Audit Methodology	Static Analysis
Key Components	

Vulnerability Summary

Vulnerability Level	Total
Critical	2
Major	0
Medium	0
Minor	0
Informational	0
Discussion	0

Findings

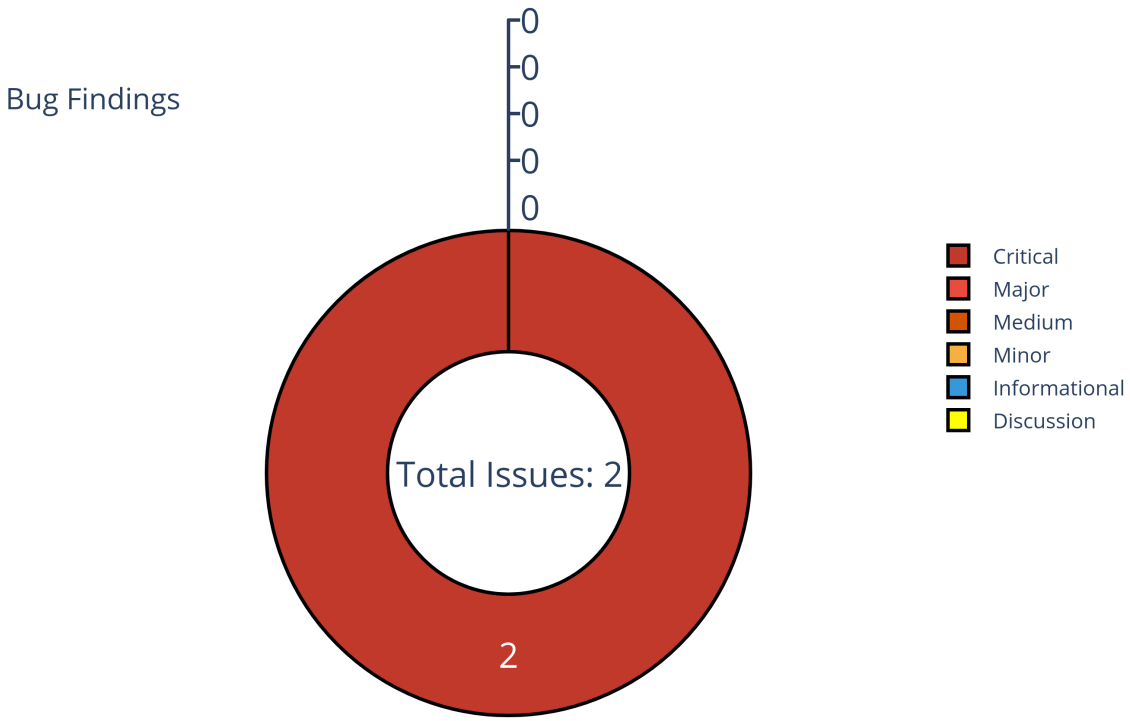

Figure 1: Findings

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

Issue: 0: MissingKeyCheck

Category	Severity	Status
MissingKeyCheck	Critical	UnResolved

- Location

```
src/processor.rs
```

- Code Context

```
94  fn process_exchange(
95          accounts: &[AccountInfo],
96          amount_expected_by_taker: u64,
97          program_id: &Pubkey,
98      ) -> ProgramResult {
99          let account_info_iter = &mut accounts.iter();
100         let taker = next_account_info(account_info_iter)?;
101
102         if !taker.is_signer {
103             return Err(ProgramError::MissingRequiredSignature);
104         }
105         let takers_sending_token_account =
            ↪  next_account_info(account_info_iter)?;
106
107         let takers_token_to_receive_account =
            ↪  next_account_info(account_info_iter)?;
108
109         let pdas_temp_token_account =
            ↪  next_account_info(account_info_iter)?;
110         let pdas_temp_token_account_info =
111             TokenAccount::unpack(&pdas_temp_token_account.data.borrow())?;
112         let (pda, nonce) = Pubkey::find_program_address(&[b"escrow"],
            ↪  program_id);
113
114         if amount_expected_by_taker != pdas_temp_token_account_info.amount
            ↪  {
115             return Err(EscrowError::ExpectedAmountMismatch.into());
```

```
116         }
117         let initializers_main_account =
            ↪  next_account_info(account_info_iter)?;
118         let initializers_token_to_receive_account =
            ↪  next_account_info(account_info_iter)?;
119         let escrow_account = next_account_info(account_info_iter)?;
120
121         let escrow_info = Escrow::unpack(&escrow_account.data.borrow())?;
122
123         if escrow_info.temp_token_account_pubkey !=
            ↪  *pdas_temp_token_account.key {
124             return Err(ProgramError::InvalidAccountData);
125         }
126         if escrow_info.initializer_pubkey != *initializers_main_account.key
            ↪  {
127             return Err(ProgramError::InvalidAccountData);
128         }
129         if escrow_info.initializer_token_to_receive_account_pubkey
130             != *initializers_token_to_receive_account.key
131         {
132             return Err(ProgramError::InvalidAccountData);
133         }
134
135         let token_program = next_account_info(account_info_iter)?;
136
137         let transfer_to_initializer_ix = spl_token::instruction::transfer(
138             token_program.key,
139             takers_sending_token_account.key,
140             initializers_token_to_receive_account.key,
141             taker.key,
142             &[&taker.key],
143             escrow_info.expected_amount,
144         )?;
145         msg!("Calling the token program to transfer tokens to the escrow's
            ↪  initializer...");
146         invoke(
147             &transfer_to_initializer_ix,
148             &[
149                 takers_sending_token_account.clone(),
150                 initializers_token_to_receive_account.clone(),
151                 taker.clone(),
152                 token_program.clone(),
```

```
            ],
        )?;

        let pda_account = next_account_info(account_info_iter)?;

        let transfer_to_taker_ix = spl_token::instruction::transfer(
            token_program.key,
            pdas_temp_token_account.key,
            takers_token_to_receive_account.key,
            &pda,
            &[&pda],
            pdas_temp_token_account_info.amount,
        )?;
        msg!("Calling the token program to transfer tokens to the
        ↪  taker...");
        invoke_signed(
            &transfer_to_taker_ix,
            &[
                pdas_temp_token_account.clone(),
                takers_token_to_receive_account.clone(),
                pda_account.clone(),
                token_program.clone(),
            ],
            &[&[&b"escrow"[..], &[nonce]]],
        )?;

        let close_pdas_temp_acc_ix = spl_token::instruction::close_account(
            token_program.key,
            pdas_temp_token_account.key,
            initializers_main_account.key,
            &pda,
            &[&pda],
        )?;
        msg!("Calling the token program to close pda's temp account...");
        invoke_signed(
            &close_pdas_temp_acc_ix,
            &[
                pdas_temp_token_account.clone(),
                initializers_main_account.clone(),
                pda_account.clone(),
                token_program.clone(),
            ],
```

```
194             &[&[&b"escrow"[..], &[nonce]]],
195         )?;
196 
197         msg!("Closing the escrow account...");
198         **initializers_main_account.lamports.borrow_mut() =
    ↪  initializers_main_account
199             .lamports()
200             .checked_add(escrow_account.lamports())
201             .ok_or(EscrowError::AmountOverflow)?;
202         **escrow_account.lamports.borrow_mut() = 0;
203 
204         Ok(())
205     }
206 
```

- Call Stack

```
1  fn entrypoint::process_instruction(){// src/entrypoint.rs:7:1: 13:2 }
2      fn processor::EscrowProcessor::process(){// src/processor.rs:16:5: 33:6
       ↪  }
3          fn processor::EscrowProcessor::process_exchange(){//
           ↪  src/processor.rs:94:5: 205:6 }
4 
```

- description:
- link:
- alleviation:

Issue: 1: MissingKeyCheck

Category	Severity	Status
MissingKeyCheck	Critical	UnResolved

- Location

src/processor.rs

- Code Context

```
35 fn process_init_escrow(
36         accounts: &[AccountInfo],
37         amount: u64,
38         program_id: &Pubkey,
39     ) -> ProgramResult {
40         let account_info_iter = &mut accounts.iter();
41         let initializer = next_account_info(account_info_iter)?;
42 
43         if !initializer.is_signer {
44             return Err(ProgramError::MissingRequiredSignature);
45         }
46         let temp_token_account = next_account_info(account_info_iter)?;
47 
48         let token_to_receive_account =
    ↪  next_account_info(account_info_iter)?;
49         if *token_to_receive_account.owner != spl_token::id() {
50             return Err(ProgramError::IncorrectProgramId);
51         }
52         let escrow_account = next_account_info(account_info_iter)?;
53         let rent =
    ↪  &Rent::from_account_info(next_account_info(account_info_iter)?)?;
54 
55         if !rent.is_exempt(escrow_account.lamports(),
    ↪  escrow_account.data_len()) {
56             return Err(EscrowError::NotRentExempt.into());
57         }
58         let mut escrow_info =
    ↪  Escrow::unpack_unchecked(&escrow_account.data.borrow())?;
```

```
59          if escrow_info.is_initialized() {
60              return Err(ProgramError::AccountAlreadyInitialized);
61          }
62
63          escrow_info.is_initialized = true;
64          escrow_info.initializer_pubkey = *initializer.key;
65          escrow_info.temp_token_account_pubkey = *temp_token_account.key;
66          escrow_info.initializer_token_to_receive_account_pubkey =
    ↪  *token_to_receive_account.key;
67          escrow_info.expected_amount = amount;
68
69          Escrow::pack(escrow_info, &mut escrow_account.data.borrow_mut())?;
70          let (pda, _nonce) = Pubkey::find_program_address(&[b"escrow"],
            ↪  program_id);
71
72          let token_program = next_account_info(account_info_iter)?;
73          let owner_change_ix = spl_token::instruction::set_authority(
74              token_program.key,
75              temp_token_account.key,
76              Some(&pda),
77              spl_token::instruction::AuthorityType::AccountOwner,
78              initializer.key,
79              &[&initializer.key],
80          )?;
81
82          msg!("Calling the token program to transfer token account
            ↪  ownership...");
83          invoke(
84              &owner_change_ix,
85              &[
86                  temp_token_account.clone(),
87                  initializer.clone(),
88                  token_program.clone(),
89              ],
90          )?;
91          Ok(())
92      }
93
```

- Call Stack

```
fn entrypoint::process_instruction(){// src/entrypoint.rs:7:1: 13:2 }
    fn processor::EscrowProcessor::process(){// src/processor.rs:16:5: 33:6
    ↪  }
        fn processor::EscrowProcessor::process_init_escrow(){//
        ↪  src/processor.rs:35:5: 92:6 }

```

- 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.