Cleandefi

Smart Contract Security Audit

Prepared by ShellBoxes

June 9th, 2022 - July 5th, 2022

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Cleandefi
Version	1.0
Classification	Public

Scope

The Cleandefi program

Files	MD5 Hash
src/entrypoint.rs	5c7b7c676dedd6b4f9c4b304589b04aa
src/error.rs	eb8c029f1ed7bb9d2cc6e7853d5806c5
src/instruction.rs	7a681c3a56d9c5ff9f0d11677852bb1d
src/lib.rs	2e74efe8cf8f9a1a9a58032eed588912
src/native_mint.rs	107bd09494956b9446fd77c905a44d7a
src/processor.rs	58a13d2535c5d0566092361223f4c0c6
src/state.rs	c3488b85aa7e8da2f25d26f0f596a4cd

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

Cleandefi engaged ShellBoxes to conduct a security assessment on the Cleandefi beginning on June 9th, 2022 and ending July 5th, 2022. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Cleandefi

The Best Swap Rates with their innovative AMM scanning module.
CleanDefi is an innovative decentralized AMM, Incubator, Yield Farming & NFT Launch solution governed by the community and powered by SOLANA.

Issuer	Cleandefi
Website	https://cleandefi.finance
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact	Likelihood: High	Likelihood: Medium	Likelihood: Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Cleandefi implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include , 1 medium-severity, 3 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
Forking SPL Token Program With No Custom Changes	MEDIUM	Acknowledged
Missing Validation In The Number Of Signers	LOW	Fixed
Missing Validation On The State Of The Delegated Account	LOW	Fixed
Incorrect Result Due To U64 Convertion	LOW	Fixed

3 Finding Details

A processor.rs

A.1 Forking SPL Token Program With No Custom Changes [MEDIUM]

Description:

SPL tokens are to Solana what ERC-20, ERC-721, and ERC-1155 tokens are to the Ethereum network. As such, SPL can be seen as the token standard for the Solana blockchain. However, the SPL token program have some limitations concerning the implementation of a custom logic since everyone uses the same deployed token program, the solution to that is to create a custom program that suits the required custom implementation separated from the SPL token program. In the case of the audited program, it is a fork of the SPL token program without any custom implementation, which will cause losing the support of the wallets to the standard SPL tokens and deploying a new program that implements functionalities that are already existing in the standard.

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

It is recommended to create a token in the SPL Token Program using the spl-token CLI, as there is no need to deploy a new program if the business logic does not require custom features that cannot be implemented using the spl-token.

Status - Acknowledged

The CleanDeFi team has acknowledged the risk, stating that it is convenient for the CleanDeFi project development in the future.

A.2 Missing Validation In The Number Of Signers [LOW]

Description:

In the InitializeMultisig instruction, n is the number of signers and m is the number of required signers, if m has a value that is greater than n, the Multisig will not be able to perform any action.

Code:

Listing 1: processor.rs

```
192 let signer_infos = account_info_iter.as_slice();
193 multisig.m = m;
194 multisig.n = signer_infos.len() as u8;
195 if !is_valid_signer_index(multisig.n as usize) {
196     return Err(TokenError::InvalidNumberOfProvidedSigners.into());
197 }
```

Recommendation:

It is recommended to add a verification in the Multisig initialization to make sure that the n that will be stored in the Multisig account is higher than m.

Status - Fixed

The CleanDeFi team has fixed the issue by requiring the n that will be stored in the Multisig account to be higher than m.

A.3 Missing Validation On The State Of The Delegated Account [LOW]

Description:

Using the process_approve function, a user can delegate a specific amount to be manipulated by another account, however in this function we are only verifying the state of the source account not the delegate account.

Code:

Listing 2: processor.rs

```
359 if source_account.is_frozen() {
360     return Err(TokenError::AccountFrozen.into());
361 }
```

Recommendation:

Consider validating also the delegate account by using the is_frozen function.

Status - Fixed

The CleanDeFi has fixed the issue by adding a condition that makes sure that the delegate account is not frozen.

A.4 Incorrect Result Due To U64 Convertion [LOW]

Description:

The ui_amount_to_amount function convert the UI representation by taking the ui_amount and multiply it by 10.pow(decimals), the result is converted to u64. Therefore, by taking an amount greater than 2.pow(64)/10.pow(9) (if the decimal is 9), the result will be incorrect due the conversion exceeds 2.pow(64).

Code:

Listing 3: processor.rs

```
21 pub fn ui_amount_to_amount(ui_amount: f64, decimals: u8) -> u64 {
22     (ui_amount * 10_usize.pow(decimals as u32) as f64) as u64}
```

Recommendation:

Consider returning an u128 or verifying that ui_amount is always less than 2.pow(64)/10.pow(9).

Status - Fixed

The CleanDeFi team has fixed the issue by returning an u128 to avoid precision errors.

4 Best Practices

BP.1 Redundant Functions In The process_initialize_mint

Description:

In the processor module, we found two public functions process_initialize_mint and process_initialize_mint2 calling the same internal function _process_initialize_mint with the parameter rent_sysvar_account different. The same remark goes for the process_initialize_account and the process_initialize_multisig.

Code:

Listing 4: processor.rs (Line 63)

```
1  //Processes an [InitializeMint](enum.TokenInstruction.html) instruction
2      pub fn process_initialize_mint(
3          accounts: &[AccountInfo],
4          decimals: u8,
5          mint_authority: Pubkey,
6          freeze_authority: COption<Pubkey>,
7      ) -> ProgramResult {
8          Self::_process_initialize_mint(accounts,decimals,mint_authority,
9          freeze_authority, true)
10     }
11
12 //Processes an [InitializeMint2](enum.TokenInstruction.html) instruction
13     pub fn process_initialize_mint2(
14         accounts: &[AccountInfo],
15         decimals: u8,
16         mint_authority: Pubkey,
17         freeze_authority: COption<Pubkey>,
18     ) -> ProgramResult {
```

```
        Self::_process_initialize_mint(accounts,decimals,mint_authority,
        freeze_authority, false)
    }
```

Listing 5: processor.rs (Line 143)

```
/// Processes an [InitializeAccount](enum.TokenInstruction.html)
/// instruction
pub fn process_initialize_account(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
) -> ProgramResult {
    Self::_process_initialize_account(program_id, accounts, None, true)
}

/// Processes an [InitializeAccount2](enum.TokenInstruction.html)
///instruction.
pub fn process_initialize_account2(
  program_id: &Pubkey,
  accounts: &[AccountInfo],
  owner: Pubkey,
) -> ProgramResult {
Self::_process_initialize_account(program_id,accounts,Some(&owner),true)
}

/// Processes an [InitializeAccount3](enum.TokenInstruction.html)
/// instruction.
pub fn process_initialize_account3(
  program_id: &Pubkey,
  accounts: &[AccountInfo],
  owner: Pubkey,
) -> ProgramResult {
Self::_process_initialize_account(program_id,accounts,Some(&owner),false)
}
```

Listing 6: processor.rs (Line 211)

```rust
/// Processes a [InitializeMultisig](enum.TokenInstruction.html)
/// instruction.
pub fn process_initialize_multisig(accounts: &[AccountInfo], m: u8)
  -> ProgramResult {
    Self::_process_initialize_multisig(accounts, m, true)
}

/// Processes a [InitializeMultisig2](enum.TokenInstruction.html)
/// instruction.
pub fn process_initialize_multisig2(accounts: &[AccountInfo], m: u8)
  -> ProgramResult {
    Self::_process_initialize_multisig(accounts, m, false)
}
```

BP.2 Duplicate Code In The Validation Of The Index Of Signers

Description:

The `_process_initialize_multisig` takes the length of the signers and verify if it's between `MIN_SIGNERS` and `MAX_SIGNERS`, however this verification is duplicated two times. Consider removing the second verification (L198).

Code:

Listing 7: KommunitasProject (Line 195)

```rust
if !is_valid_signer_index(multisig.n as usize) {
    return Err(TokenError::InvalidNumberOfProvidedSigners.into());
}
if !is_valid_signer_index(multisig.m as usize) {
    return Err(TokenError::InvalidNumberOfRequiredSigners.into());
}
```

5 Static Analysis

5.1 Soteria

Description:

Soteria can automatically detect security vulnerabilities in Solana programs by checking all code paths against common pitfalls.The fundamental idea is to examine the data flow of each user account supplied to the program and flag it as untrustworthy if its validity is not properly verified in the program's execution context.

Results:

```
Analyzing /program/.coderrect/build/bpfel-unknown-unknown/release/all.ll ...
 -  [00m:01s] Loading IR From File
 -  [00m:00s] Running Compiler Optimization Passes
EntryPoints:
entrypoint
 -  [00m:00s] Running Compiler Optimization Passes
 -  [00m:00s] Running Pointer Analysis
 -  [00m:00s] Building Static Happens-Before Graph
 -  [00m:00s] Detecting Vulnerabilities
detected 0 untrustful accounts in total.
detected 0 unsafe math operations in total.
--------The summary of potential vulnerabilities in all.ll--------
       No vulnerabilities detected
```

5.2 Cargo Tarpaulin

Results:

```
INFO cargo_tarpaulin::report: Coverage Results:
|| Tested/Total Lines:
|| src/entrypoint.rs: 0/5
|| src/error.rs: 2/4
```

```
|| src/instruction.rs: 652/713
|| src/lib.rs: 34/36
|| src/native_mint.rs: 7/7
|| src/processor.rs: 2836/2892
|| src/state.rs: 236/236
|| tests/action.rs: 0/5
||
96.64% coverage, 3767/3898 lines covered
```

5.3 Tests

Results:

```
running 47 tests
test native_mint::test_id ... ok
test instruction::test::test_instruction_packing ... ok
test native_mint::tests::test_decimals ... ok
test processor::tests::test_amount_to_ui_amount ... ok
test processor::tests::test_approve ... ok
test processor::tests::test_approve_dups ... ok
test processor::tests::test_burn ... ok
test processor::tests::test_burn_dups ... ok
test processor::tests::test_close_account ... ok
test processor::tests::test_close_authority_close_account_dups ... ok
test processor::tests::test_error_unwrap - should panic ... ok
test processor::tests::test_freeze_account ... ok
test processor::tests::test_freeze_thaw_dups ... ok
test processor::tests::test_frozen ... ok
test processor::tests::test_get_account_data_size ... ok
test processor::tests::test_burn_and_close_system_and_incinerator_tokens
... ok
test processor::tests::test_initialize_account2_and_3 ... ok
test processor::tests::test_initialize_immutable_owner ... ok
test processor::tests::test_initialize_mint2 ... ok
test processor::tests::test_initialize_mint ... ok
```

```
test processor::tests::test_initialize_mint_account ... ok
test processor::tests::test_mint_to ... ok
test processor::tests::test_mintable_token_with_zero_supply ... ok
test processor::tests::test_mint_to_dups ... ok
test processor::tests::test_multisig ... ok
test processor::tests::test_overflow ... ok
test processor::tests::test_native_token ... ok
test processor::tests::test_owner_close_account_dups ... ok
test processor::tests::test_pack_unpack ... ok
test processor::tests::test_print_error ... ok
test processor::tests::test_self_transfer ... ok
test processor::tests::test_set_authority_dups ... ok
test processor::tests::test_set_authority ... ok
test processor::tests::test_sync_native ... ok
test processor::tests::test_transfer ... ok
test processor::tests::test_transfer_dups ... ok
test processor::tests::test_ui_amount_to_amount ... ok
test processor::tests::test_unique_account_sizes ... ok
test processor::tests::test_validate_owner ... ok
test state::tests::test_account_state ... ok
test state::tests::test_mint_unpack_from_slice ... ok
test state::tests::test_multisig_unpack_from_slice ... ok
test state::tests::test_unpack_coption_key ... ok
test state::tests::test_unpack_coption_u64 ... ok
test state::tests::test_unpack_token_mint ... ok
test state::tests::test_unpack_token_owner ... ok
test test_id ... ok
test result: ok. 47 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out;
finished in 0.01s
    Running tests/action.rs (target/debug/deps/action-aa7da80bc993f1d6)
running 0 tests
test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out;
finished in 0.00s
    Running tests/assert_instruction_count.rs (target/debug/deps/assert_
```

```
instruction_count-c329597181c539d9)
running 0 tests
test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out;
finished in 0.00s
   Doc-tests spl-token
running 0 tests
test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out;
finished in 0.00s
```

Conclusion:

Most of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

6 Conclusion

In this audit, we examined the design and implementation of Cleandefi contract and discovered several issues of varying severity. Cleandefi team addressed 3 issues raised in the initial report and implemented the necessary fixes, while acknowledging the issue 'A1'. Shellboxes' auditors advised Cleandefi Team to maintain a high level of vigilance and to keep this issue in mind in order to avoid any future complications.

For a Contract Audit, contact us at contact@shellboxes.com