F Y E O

Security Code Review of Voltr Vault

Voltr

March 2025
Version 1.0

Presented by:
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Public

Table of Contents

Executive Summary....2
- Overview....2
- Key Findings....2
- Scope and Rules of Engagement....3

Technical Analyses and Findings....5
- Findings....6
- Technical Analysis....6
- Conclusion....6

Technical Findings....7
- General Observations....7
- Unused harvest_fee() function results in unclaimed accumulated fees....8
- Protocol fees are not accumulated....9
- Amount not checked to be positive....11
- Dead code....12
- Incomplete test coverage....13
- Insecure initialization....14
- LP token does not allow Token2022....15
- Potential panic when copying strings to fixed arrays during vault initialization....16

Our Process....16
- Methodology....17
- Kickoff....17
- Ramp-up....17
- Review....18
- Code Safety....18
- Technical Specification Matching....18
- Reporting....19
- Verify....19

Additional Note....19

The Classification of vulnerabilities....20

Executive Summary

Overview

Voltr engaged FYEO Inc. to perform a Security Code Review of Voltr Vault.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on March 10 - March 15, 2025, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following issues have been identified during the testing period. These should be prioritized for remediation to reduce the risk they pose:

- FYEO-VOLTR-01 – Unused harvest_fee() function results in unclaimed accumulated fees
- FYEO-VOLTR-02 – Protocol fees are not accumulated
- FYEO-VOLTR-03 – Amount not checked to be positive
- FYEO-VOLTR-04 – Dead code
- FYEO-VOLTR-05 – Incomplete test coverage
- FYEO-VOLTR-06 – Insecure initialization
- FYEO-VOLTR-07 – LP token does not allow Token2022
- FYEO-VOLTR-08 – Potential panic when copying strings to fixed arrays during vault initialization

Based on our review process, we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

The FYEO Review Team performed a Security Code Review of Voltr Vault. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a private repository at https://github.com/voltrxyz/vault-program with the commit hash e2cd60bb4e4406b8f312d843452e5247c35976a2.

Remediations have been submitted with the commit hash a53f14fec9ca8efa5fa147a2059d12c48b5d11ff.

Files included in the code review

```
vault-program/
└── programs/
    └── voltr-vault/
        └── src/
            ├── instructions/
            │   ├── add_adaptor.rs
            │   ├── cancel_request_withdraw_vault.rs
            │   ├── deposit_strategy.rs
            │   ├── deposit_vault.rs
            │   ├── direct_withdraw_strategy.rs
            │   ├── harvest_fee.rs
            │   ├── init_or_update_protocol.rs
            │   ├── initialize_direct_withdraw_strategy.rs
            │   ├── initialize_strategy.rs
            │   ├── initialize_vault.rs
            │   ├── mod.rs
            │   ├── remove_adaptor.rs
            │   ├── request_withdraw_vault.rs
            │   ├── update_vault.rs
            │   ├── withdraw_strategy.rs
            │   └── withdraw_vault.rs
            ├── state/
            │   ├── adaptor_add_receipt.rs
            │   ├── direct_withdraw_init_receipt.rs
            │   ├── mod.rs
            │   ├── protocol.rs
            │   ├── request_withdraw_vault_receipt.rs
            │   ├── strategy_init_receipt.rs
            │   └── vault.rs
            ├── utils/
            │   ├── accounting.rs
            │   ├── constants.rs
```

Files included in the code review
│ ├── decimal.rs │ ├── helpers.rs │ ├── math.rs │ └── mod.rs ├── error.rs └── lib.rs

Table 1: Scope

Technical Analyses and Findings

During the Security Code Review of Voltr Vault, we discovered:

- 1 finding with HIGH severity rating.
- 1 finding with MEDIUM severity rating.
- 6 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

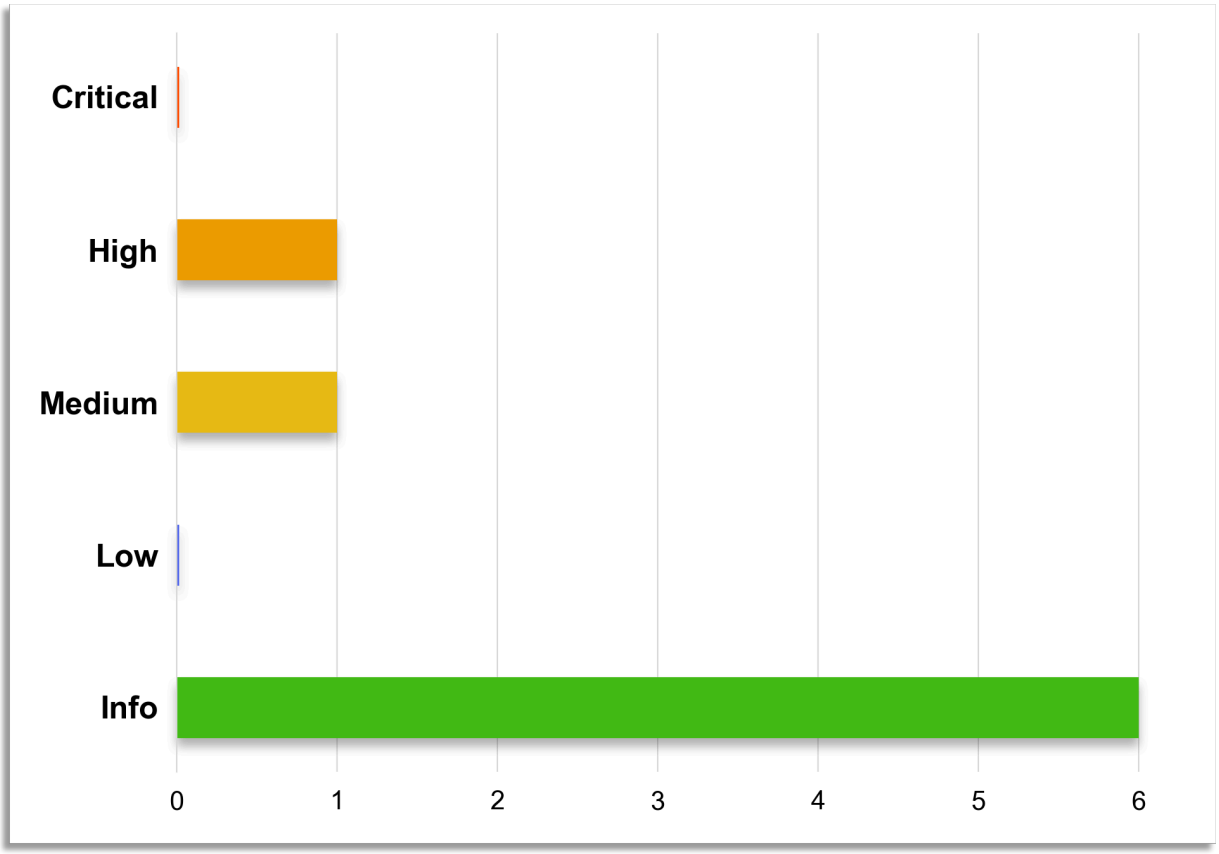

Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Severity	Description
FYEO-VOLTR-01	High	Unused harvest_fee() function results in unclaimed accumulated fees
FYEO-VOLTR-02	Medium	Protocol fees are not accumulated
FYEO-VOLTR-03	Informational	Amount not checked to be positive
FYEO-VOLTR-04	Informational	Dead code
FYEO-VOLTR-05	Informational	Incomplete test coverage
FYEO-VOLTR-06	Informational	Insecure initialization
FYEO-VOLTR-07	Informational	LP token does not allow Token2022
FYEO-VOLTR-08	Informational	Potential panic when copying strings to fixed arrays during vault initialization

Table 2: Findings Overview

Technical Analysis

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

Conclusion

Based on our review process, we conclude that the code implements the documented functionality to the extent of the reviewed code.

Technical Findings

General Observations

The `VoltrVault` program presents a modular and flexible architecture for managing tokenized vaults on Solana. It enables the creation and management of vaults with configurable parameters, including deposit/withdrawal operations, strategy integration, adaptor management, and fee collection. The design focuses on modularity, extensibility, and precision in asset management, employing robust mathematical utilities (SafeMath, DecimalExtra) to ensure accuracy in financial calculations.

The protocol supports complex workflows such as strategy deposits/withdrawals, direct withdrawals via adaptors, and dynamic vault configurations. Role-based access control is well-integrated, separating the responsibilities of managers, admins, and users. Additionally, the architecture includes features like high-water mark updates, withdrawal request handling with waiting periods, and performance/management fee calculations, reflecting an advanced understanding of DeFi mechanics.

The system enforces strict operational state control using OperationalState flags, ensuring that protocol operations are enabled/disabled based on predefined states.

However, several areas require attention and improvement to ensure robustness, reliability, and operational completeness: - `harvest_fee()` function, responsible for minting and distributing accumulated protocol, admin, and manager fees, is defined but never invoked within the system - although the vault tracks `accumulated_lp_protocol_fees` within its state, there is no logic in the codebase that updates this value during vault operations - incomplete test coverage.

Unused harvest_fee() function results in unclaimed accumulated fees

Finding ID: FYEO-VOLTR-01
Severity: **High**
Status: **Remediated**

Description

The `harvest_fee()` handler is implemented but not exposed through any public instruction in the `voltr_vault` program. As a result, the accumulated fees recorded in the Vault's `fee_state` (specifically `accumulated_lp_manager_fees`, `accumulated_lp_admin_fees`, and `accumulated_lp_protocol_fees`) cannot be harvested or distributed.

Proof of Issue

File name: programs/voltr-vault/src/lib.rs
Line number: 12

```
#[program]
pub mod voltr_vault {
    use super::*;
    ...
    // No `pub fn harvest_fee(...)` exists
    ...
}
```

This means no entity can trigger fee distribution within the current deployed logic.

Severity and Impact Summary

- Without the execution of harvest_fee, these fees:
- Remain virtual and are never converted into actual LP tokens.
- Managers, administrators cannot claim their rewards.
- This breaks the economic incentives model, as managers and admins expect to receive their accumulated fees in LP tokens.

Recommendation

Expose `harvest_fee` as a public instruction in the `voltr_vault` program.

Protocol fees are not accumulated

Finding ID: FYEO-VOLTR-02
Severity: Medium
Status: Remediated

Description

The protocol fee mechanism (`accumulated_lp_protocol_fees`) is declared in the Vault contract as part of the `FeeState` structure but is never updated or distributed within the current implementation of the `voltr_vault` program.

In particular:

- The function `handle_performance_fee_and_update_vault()` updates only `accumulated_lp_admin_fees` and `accumulated_lp_manager_fees`.
- There is no logic in any of the fee handling functions or harvest_fee to increment or mint LP tokens for the protocol.
- As a result, the protocol will not receive its expected share of accumulated fees, which may violate the intended economic model.

Proof of Issue

File name: programs/voltr-vault/src/state/vault.rs
Line number: 263

Declaration:

```
pub struct FeeState {
    /// The accumulated manager fees in the vault.
    pub accumulated_lp_manager_fees: u64,

    /// The accumulated admin fees in the vault.
    pub accumulated_lp_admin_fees: u64,

    /// The accumulated protocol fees in the vault.
    pub accumulated_lp_protocol_fees: u64,

    /// Reserved bytes for future use.
    pub reserved: [u8; 24],
}
```

File name: programs/voltr-vault/src/state/vault.rs
Line number: 439

No mention of updating `accumulated_lp_protocol_fees`:

```
    pub fn handle_performance_fee_and_update_vault<'info>(
        &mut self,
        is_profit: bool,// true
        pnl_amount: u64,//50
        vault_asset_mint_decimals: u8,//18
        vault_lp_mint_supply: u64,//300
        vault_lp_mint_decimals: u8//9
```

```
    ) -> Result<()> {
      ...
      if lp_tokens_to_mint > 0 {
         let (admin_fee_amount, manager_fee_amount) =
            self.calc_fee_split(lp_tokens_to_mint)?;
         self.increment_admin_fees(admin_fee_amount)?;
         self.increment_manager_fees(manager_fee_amount)?;
      }
      ...
    }
```

File name: programs/voltr-vault/src/instructions/harvest_fee.rs
Line number: 70

Only `admin` and `manager` fees are minted:

```
pub fn harvest_fee(ctx: Context<HarvestFee>) -> Result<()> {
    ctx.accounts.protocol.load()?.check_operation_state(OperationalState::HARVEST_FEE)?;
    ctx.accounts.mint_fee_to_admin()?;
    ctx.accounts.mint_fee_to_manager()?;
    ctx.accounts.vault.load_mut()?.reset_fee_state()?;

    Ok(())
}
```

Severity and Impact Summary

This issue does not introduce a security vulnerability or risk to user funds, it affects the protocol's ability to earn fees. If protocol fees are part of the expected economic model or governance incentives, their absence can impact protocol sustainability and fairness.

Recommendation

1. Increment `accumulated_lp_protocol_fees` where appropriate, such as inside handle_performance_fee_and_update_vault or other fee distribution functions.
2. Update the `harvest_fee()` function to mint protocol fees to a designated protocol address (currently missing from the context).

Amount not checked to be positive

Finding ID: FYEO-VOLTR-03
Severity: Informational
Status: Acknowledged

Description

The deposit and withdrawal functions do not require a positive amount. They still update `vault.update_last_updated_ts()`.

Proof of Issue

File name: programs/voltr-vault/src/instructions/deposit_vault.rs
Line number: 25

```
pub fn handler_deposit_vault(ctx: Context<DepositVault>, amount: u64) -> Result<()> {
```

File name: programs/voltr-vault/src/instructions/request_withdraw_vault.rs
Line number: 20

```
pub fn handler_request_withdraw_vault(
    ctx: Context<RequestWithdrawVault>,
    amount: u64,
    is_amount_in_lp: bool,
    is_withdraw_all: bool
) -> Result<()> {
```

Severity and Impact Summary

Not a security concern.

Recommendation

Check the amount given is positive.

Dead code

Finding ID: FYEO-VOLTR-04
Severity: **Informational**
Status: **Remediated**

Description

Below are code snippets that are not used anywhere.

Proof of Issue

File name: programs/voltr-vault/src/state/adaptor_add_receipt.rs
Line number: 21

```
pub struct AdaptorAddReceipt {
    ...
    pub is_active: bool,
    ...
}
```

The `is_active` field is not initialized explicitly (left `false` by default) and the value is not changed or checked anywhere else in the code.

File name: programs/voltr-vault/src/error.rs
Line number: 4

```
pub enum VaultError {
    ...
    #[msg("Invalid account owner.")]
    InvalidAccountOwner,
    #[msg("Already initialized.")]
    AlreadyInitialized,
    ...
    #[msg("Not rent exempt.")]
    NotRentExempt,
    ...
    #[msg("Strategy not empty.")]
    StrategyNotEmpty,
}
```

Several errors in the `VaultError` are never used

Severity and Impact Summary

This does not carry any threat.

Recommendation

If a part of the code is not used in the contract it would be best to remove it.

Incomplete test coverage

Finding ID: FYEO-VOLTR-05
Severity: Informational
Status: Acknowledged

Description

The current test coverage is not complete and covers only basic scenarios of working with `Vault`:

- Creation
- Deposits
- Withdrawals
- Configuration updates.

Proof of Issue

There are no tests for interaction with strategies:

- Initialize_strategy
- Deposit_strategy
- Withdraw_strategy

No tests for adapters: - add_adaptor - remove_adaptor

There are also no tests for adapters for collecting and distributing commissions via `harvest_fee()`.

Negative tests for access rights, limits (maxCap, fees) are also not implemented.

Severity and Impact Summary

A lack of tests can erode the security of the project.

Recommendation

Improve the test coverage.

Insecure initialization

Finding ID: FYEO-VOLTR-06
Severity: Informational
Status: Acknowledged

Description

The initialization is done using a public function. There is a risk someone else can call it first. Also the `new_admin` did not sign which may lead to a loss of access if mistakes are made.

Proof of Issue

File name: programs/voltr-vault/src/instructions/init_or_update_protocol.rs
Line number: 15

```
pub struct InitOrUpdateProtocol<'info> {
    #[account(mut)]
    pub payer: Signer<'info>,

    #[account()]
    pub current_admin: Signer<'info>,

    /// CHECK: new admin
    #[account()]
    pub new_admin: AccountInfo<'info>,

    #[account(
        init_if_needed,
        seeds = [PROTOCOL_SEED],
        bump,
        payer = payer,
        space = std::mem::size_of::<Protocol>() + 8
    )]
    pub protocol: AccountLoader<'info, Protocol>,
```

Severity and Impact Summary

Anyone can initialize this program.

Recommendation

For upgradeable programs, it is recommended to use the following pattern to initialize the program.

```
pub owner: Signer<'info>,

#[account(constraint = program.programdata_address()? == Some(program_data.key()))]
pub program: Program<'info, MyProgram>,

#[account(constraint = program_data.upgrade_authority_address == Some(owner.key()))]
pub program_data: Account<'info, ProgramData>,
```

Also consider using a two step process to transfer ownership by proposing a new admin, and having the new admin accept their role. This would help avoid potential mistakes.

LP token does not allow Token2022

Finding ID: FYEO-VOLTR-07
Severity: Informational
Status: Acknowledged

Description

The program enforces the use of the original `Tokenkeg` program and does not allow for Token2022 for the LP tokens. This means that there is no support for token extensions such as `ImmutableOwner` which makes the associated token accounts susceptible to scamming attacks.

Proof of Issue

```
pub lp_token_program: Program<'info, Token>,

/// The user's LP ATA where we will mint LP tokens.
#[account(
    mut,
    associated_token::mint = vault_lp_mint,
    associated_token::authority = user_transfer_authority,
    associated_token::token_program = lp_token_program,
)]
pub user_lp_ata: Account<'info, TokenkegAccount>,
```

Since this uses `Tokenkeg`, there is no guarantee this account is owned by `user_transfer_authority`. While the user does sign this transaction they may not necessarily be aware of technical details such as associated token accounts and token authorities.

Severity and Impact Summary

Not supporting Token2022 means that associated token accounts will be created without `ImmutableOwner`. Which in turn implies they are susceptible to scamming attacks.

Recommendation

Consider using Token2022.

References

https://solana.com/developers/guides/token-extensions/immutable-owner
https://docs.rs/anchor-lang/latest/anchor_lang/derive.Accounts.html
(see TokenInterface section)

Potential panic when copying strings to fixed arrays during vault initialization

Finding ID: FYEO-VOLTR-08
Severity: **Informational**
Status: **Remediated**

Description

When initializing the vault, the code copies the name and description strings into fixed-size byte arrays without checking if the strings exceed the array capacity. This can lead to a runtime panic if the input strings are longer than the allocated buffers.

Proof of Issue

File name: programs/voltr-vault/src/instructions/initialize_vault.rs
Line number: 41

```
pub fn handler_initialize_vault(
    ctx: Context<InitializeVault>,
    config: VaultInitializationInput,
    name: String,
    description: String
) -> Result<()> {
    ...
    let mut name_data = [0u8; 32];
    name_data[..name.as_bytes().len()].copy_from_slice(name.as_bytes());

    let mut description_data = [0u8; 64];

description_data[..description.as_bytes().len()].copy_from_slice(description.as_bytes())
;
    ...
}
```

Severity and Impact Summary

This issue is not a vulnerability, but can cause transactions to fail during storage initialization if unexpectedly long strings are provided.

Recommendation

Modify the code to safely copy strings by limiting the number of bytes copied to the size of the destination array. For example, use the min function to determine the copy length:

```
    let mut name_data = [0u8; 32];
    let name_bytes = name.as_bytes();
    let copy_len = name_bytes.len().min(name_data.len());
    name_data[..copy_len].copy_from_slice(&name_bytes[..copy_len]);
```

Our Process

Methodology

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

Figure 2: Methodology Flow

Kickoff

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.

The Classification of vulnerabilities

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations