FYEO

Secure Code Review of Amulet Protocol

MTR Labs Pte Ltd.

July 2023
Version 1.0

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

Security Level
Public

TABLE OF CONTENTS

List of Figures

List of Tables

EXECUTIVE SUMMARY

Overview

MTR Labs Pte Ltd. engaged FYEO Inc. to perform a Secure Code Review of Amulet Protocol.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on June 12 - June 23, 2023, 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-AMULET-ID-01 – The mint of the treasury and pool accounts are not verified when used from an admin function.
- FYEO-AMULET-ID-02 – Unchecked LP Mints Supply in CreateVaultMetadataState when used from an admin function.
- FYEO-AMULET-ID-03 – Unchecked Oracle Account in CreateVaultMetadataState when used from an admin function
- FYEO-AMULET-ID-04 – Admin can deny withdrawal of funds by pausing vaults
- FYEO-AMULET-ID-05 – Hardcoded Program Initializer Authority
- FYEO-AMULET-ID-06 – LP mints can be the same account
- FYEO-AMULET-ID-07 – Lack of Input Validation in CreateVault
- FYEO-AMULET-ID-08 – New admin is not co-signer in UpdateAdmin

- FYEO-AMULET-ID-09 – The authority on the LP mint accounts is not checked
- FYEO-AMULET-ID-10 – The vault pool and treasury token accounts do not check the delegate
- FYEO-AMULET-ID-11 – Unchecked token account balance for vault pool
- FYEO-AMULET-ID-12 – Duplicate code execution in AttemptToTrigger
- FYEO-AMULET-ID-13 – In the CreateVault instruction the vault admin should co-sign
- FYEO-AMULET-ID-14 – Math inconsistency
- FYEO-AMULET-ID-15 – There is no way to clean up old accounts
- FYEO-AMULET-ID-16 – Unnecessary constraints
- FYEO-AMULET-ID-17 – Withdraw functions do not check token amount against withdraw amount

Based on account relationship graph analysis and our review process, we conclude that the reviewed code implements the documented functionality.

The re-review has verified that the findings above Informational have all been remediated.

Scope and Rules of Engagement

The FYEO Review Team performed a Secure Code Review of Amulet Protocol. 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/Amulet-Protocol/vault-programs with the commit hash e182a0ecf4e2f70d701f21a153a76eb84c26bc29.

The re-review was done on the commit hash 59884f527a94e461438e49bd8fe305a50fc6f752

Files included in the code review

```
amulet/
├── programs/
│   └── vault/
│       ├── src/
│       │   ├── admins/
│       │   │   ├── admin_update_admin_auth_info.rs
│       │   │   ├── admin_update_pause_info.rs
│       │   │   └── mod.rs
│       │   ├── instructions/
│       │   │   ├── attempt_to_trigger.rs
│       │   │   ├── create_vault_metadata_state.rs
│       │   │   ├── deposit_hedge_fund.rs
```

Files included in the code review
```
│       │    │    ├── deposit_risk_fund.rs
│       │    │    ├── mod.rs
│       │    │    ├── settle_vault.rs
│       │    │    ├── withdraw_hedge_fund.rs
│       │    │    └── withdraw_risk_fund.rs
│       │    ├── states/
│       │    │    ├── metadata_state.rs
│       │    │    └── mod.rs
│       │    ├── utils/
│       │    │    ├── calc.rs
│       │    │    └── mod.rs
│       │    ├── error.rs
│       │    └── lib.rs
│       ├── Cargo.toml
│       └── Xargo.toml
└── Cargo.toml
```

Table 1: Scope

TECHNICAL ANALYSES AND FINDINGS

During the Secure Code Review of Amulet Protocol, we discovered:

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

The following chart displays the findings by severity.

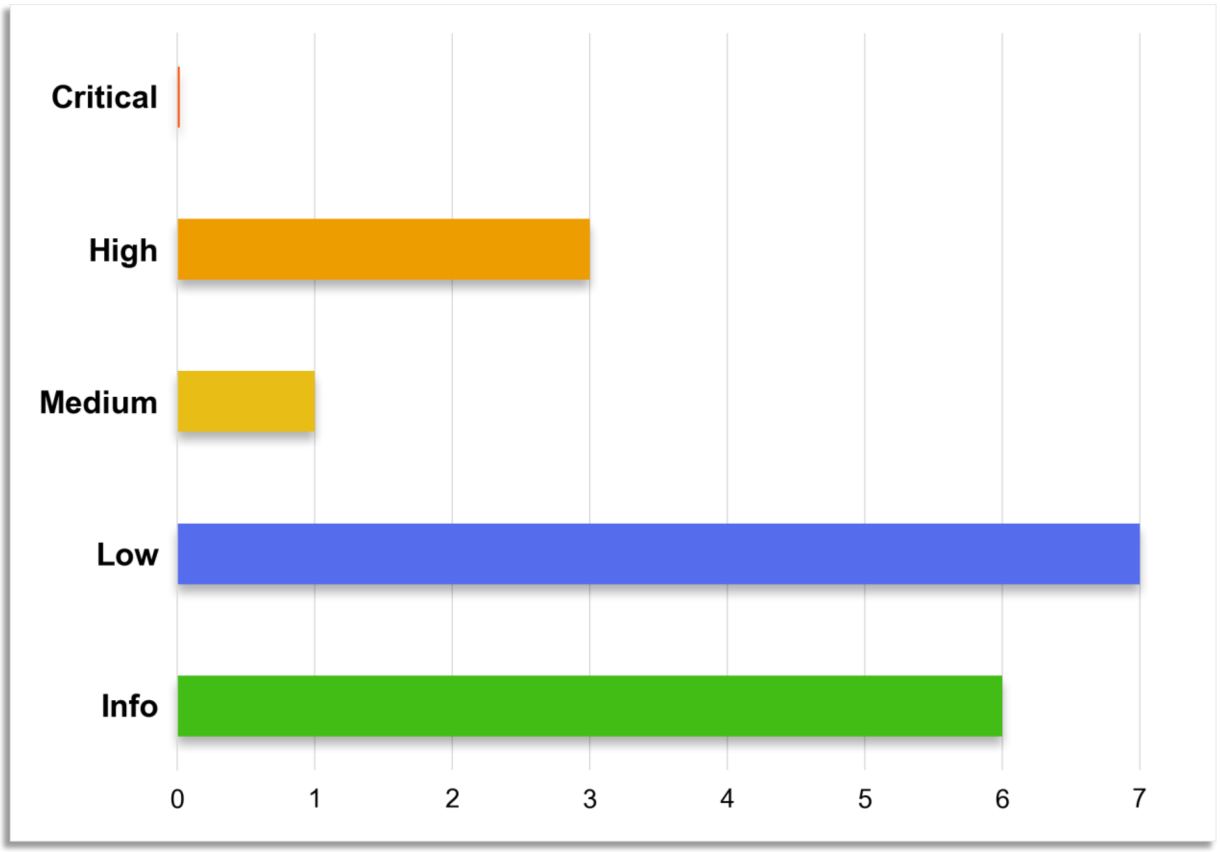


Figure 1: Findings by Severity

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-AMULET-ID-01	**High**	The mint of the treasury and pool accounts are not verified when used from an admin function.
FYEO-AMULET-ID-02	**High**	Unchecked LP Mints Supply in CreateVaultMetadataState when used from an admin function.
FYEO-AMULET-ID-03	**High**	Unchecked Oracle Account in CreateVaultMetadataState when used from an admin function
FYEO-AMULET-ID-04	**Medium**	Admin can deny withdrawal of funds by pausing vaults
FYEO-AMULET-ID-05	**Low**	Hardcoded Program Initializer Authority
FYEO-AMULET-ID-06	**Low**	LP mints can be the same account
FYEO-AMULET-ID-07	**Low**	Lack of Input Validation in CreateVault
FYEO-AMULET-ID-08	**Low**	New admin is not co-signer in UpdateAdmin
FYEO-AMULET-ID-09	**Low**	The authority on the LP mint accounts is not checked
FYEO-AMULET-ID-10	**Low**	The vault pool and treasury token accounts do not check the delegate
FYEO-AMULET-ID-11	**Low**	Unchecked token account balance for vault pool
FYEO-AMULET-ID-12	**Informational**	Duplicate code execution in AttemptToTrigger
FYEO-AMULET-ID-13	**Informational**	In the CreateVault instruction the vault admin should co-sign
FYEO-AMULET-ID-14	**Informational**	Math inconsistency
FYEO-AMULET-ID-15	**Informational**	There is no way to clean up old accounts
FYEO-AMULET-ID-16	**Informational**	Unnecessary constraints
FYEO-AMULET-ID-17	**Informational**	Withdraw functions do not check token amount against withdraw amount

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.

AUTHORIZATION

The review used relationship graphs to show the relations between account input passed to the instructions of the program. The relations are used to verify if the authorization is sufficient for invoking each instruction. The graphs show if any unreferenced accounts exist. Accounts that are not referred to by trusted accounts can be replaced by any account of an attacker's choosing and thus pose a security risk.

In particular, the graphs will show if signing accounts are referred to. If a signing account is not referred to then any account can be used to sign the transaction causing insufficient authorization.

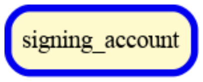
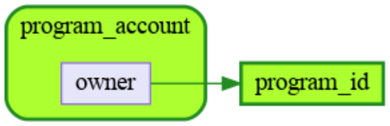
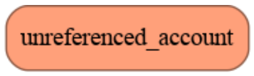
NO INSUFFICIENT AUTHORIZATION WAS FOUND BASED ON THE ANALYZES OF THE RELATIONSHIP GRAPHS. FOR DETAILS, SEE SECTION RELATIONSHIP GRAPHS

A relationship graph shows relational requirements to the input of an instruction. Notice, that it does not show outcome of the instruction.

Relationship graphs are used to analyze insufficient authorization and unchecked account relations. Insufficient authorization allows unintended access. Unchecked account relations may allow account and data injection resulting in unintended behavior and access.

Various styles are used to highlight special properties. Accounts are shown as boxes with round corners. An account box may contain smaller boxes indicating relevant account data.

The following table shows the basic styles for the relationship graphs.

	Round boxes with thick blue borders indicate accounts required to sign the transaction.
	Green round boxes highlight accounts required to be owned by the program itself.
	Red round boxes indicate accounts where ownership is not validated. Further analysis should be made to ensure this does not allow account injection attacks.

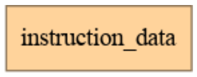
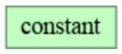
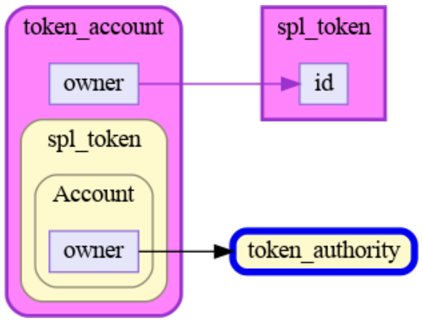
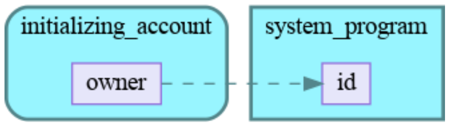
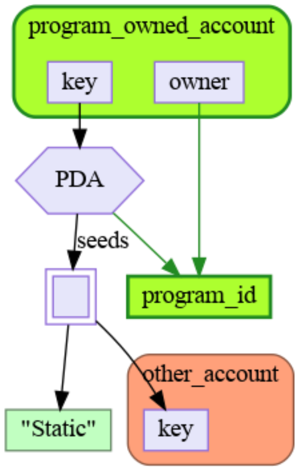
	Orange boxes indicate instruction data. As instruction data does not originate from the blockchain, it may open for data injection attacks. Caution must be taken if used for account validation.
	Green boxes indicate constants. It may refer to either hardcoded values or library code.
 token_account owner spl_token Account owner spl_token id token_authority	Inner boxes are used to indicate data structures to ease readability. Additional colors may also be used to highlight account ownership from different programs.
	Dashed lines indicate implicitly required relations. For example, relations required by another program during a cross program invocation. This is emphasized as the program cannot guarantee the behavior of external programs.
 program_owned_account key owner PDA seeds program_id other_account key "Static"	Program derived addresses are calculated using the `find_program_address` or `create_program_address` functions. To illustrate the relations to the input used for the PDA calculation, a diamond shaped box is used to show the PDA and a double-edged box to gather the seeds. As seed injection may lead to account injection, it is important that all input to the PDA is verified.

Table 3: Legend for relationship graphs starting on page 30.

Conclusion

Based on account relationship graph analysis and our review process, we conclude that the code implements the documented functionality to the extent of the reviewed code.

The re-review has verified that the findings above Informational have all been remediated.

Technical Findings

General Observations

During the code review, it was evident that the Rust code exhibited exceptional quality and structure. One notable aspect was the implementation of checked arithmetic operations to protect against potential overflows and underflows, showcasing a strong commitment to writing secure programs. The code documentation was also commendable, with the inclusion of required doc comments for unchecked accounts, providing insightful explanations behind key decisions made during development.

The utilization of the Anchor framework in this Solana project proved to be a solid foundation. The built-in account verification functionality offered by Anchor played a crucial role in ensuring the project's integrity and security. The code heavily relied on Anchor's validation and access restriction macros, further reinforcing the project's robustness.

While the code is well-constructed, a few account checks were deemed somewhat verbose during the review. For instance, token transfers between accounts of different mints are destined to fail, making various token and mint account checks unnecessary. The project demonstrated good practice by incorporating custom errors to enhance the user experience, providing informative messages to users in case of transaction failures.

The issue FYEO-AMULET-ID-05 regarding Hardcoded Program Initializer Authority will be solved by using multisig authority from Squads (serum-multisig) as the PDA for the Program Initializer Authority. This plan makes us sufficiently assured to mark the issue as remediated.

Overall, the code review yielded positive results, highlighting the exceptional craftsmanship and meticulous attention to detail evident in this Solana project. The combination of secure coding practices, intelligent utilization of the Anchor framework, and efforts to improve user experience through custom errors demonstrate expertise and ability to create great smart contracts for Solana.

THE MINT OF THE TREASURY AND POOL ACCOUNTS ARE NOT VERIFIED WHEN USED FROM AN ADMIN FUNCTION.

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

Description

The mint of neither the `vault_treasury_token_account` nor `vault_pool_token_account` token accounts are checked.

Proof of Issue

File name: lib.rs
Line number: 157

```
#[account(
    constraint = *vault_admin_auth.key == vault_treasury_token_account.owner
@ErrorCode::ConstraintInvalidTAOwner,
    // owner = anchor_spl::token::ID @ErrorCode::OwnerTokenProgramID
)]
pub vault_treasury_token_account: Account<'info, TokenAccount>,
```

File name: lib.rs
Line number: 163

```
#[account(
    constraint = *vault_auth_pda.key == vault_pool_token_account.owner
@ErrorCode::ConstraintInvalidTAOwner,
    // owner = anchor_spl::token::ID @ErrorCode::OwnerTokenProgramID
)]
pub vault_pool_token_account: Account<'info, TokenAccount>,
```

Severity and Impact Summary

Since there is no check of the mint, it is possible that the `vault_treasury_token_account` account is of a different mint than `vault_pool_token_account` and / or `vault_token_mint`. If so, this vault ends up in a DoS situation where it is impossible for depositors to withdraw funds as a transfer between token accounts of different mints can not possibly work. The code below would fail in such a situation, resulting in user funds becoming stuck.

```
.withdraw_hedge_fund(
    metadata_state_pubkey,
    &self.vault_auth_pda,
    &self.vault_pool_token_account.to_account_info(),
    &self.vault_treasury_token_account.to_account_info(),
    fee_amount,
    &self.token_program,
)
```

If the mint of `vault_pool_token_account` mismatches, it is less of an issue, as deposits would not work.

Recommendation

Make sure that `vault_treasury_token_account` and `vault_pool_token_account` are token accounts of the mint account `vault_token_mint`.

Unchecked LP Mints Supply in CreateVaultMetadataState when used from an admin function.

Finding ID: FYEO-AMULET-ID-02
Severity: **High**
Status: **Remediated**

Description

In the `CreateVaultMetadataState` instruction, there is no check to ensure that the LP mints `vault_risk_pool_lp_mint` and `vault_hedge_pool_lp_mint` have zero supply. This may lead to unexpected behavior as someone may initialize the vault with LP mints that have already been used.

Proof of Issue

File name: lib.rs
Line number: 169

```
#[account(
    constraint = vault_risk_pool_lp_mint.freeze_authority.is_none()
@ErrorCode::ConstraintMintFreezeAuthNone,
    // owner = anchor_spl::token::ID @ErrorCode::OwnerTokenProgramID
)]
pub vault_risk_pool_lp_mint: Account<'info, Mint>,

#[account(
    constraint = vault_hedge_pool_lp_mint.freeze_authority.is_none()
@ErrorCode::ConstraintMintFreezeAuthNone,
    // owner = anchor_spl::token::ID @ErrorCode::OwnerTokenProgramID
)]
pub vault_hedge_pool_lp_mint: Account<'info, Mint>,
```

Severity and Impact Summary

A malicious or inattentive admin may initialize the vault with LP mints that have already been used, this could enable somebody already holding these tokens to steal from users of this vault.

Recommendation

Add constraints to check if the LP mints have zero supply. Otherwise, these accounts could be setup as `init` and could be constructed as PDAs. This way they would be new accounts initialized by Anchor and would therefore be empty.

```
#[account(
    constraint = vault_risk_pool_lp_mint.supply == 0
@ErrorCode::ConstraintMintZeroSupply,
)]
pub vault_risk_pool_lp_mint: Account<'info, Mint>;

#[account(
    constraint = vault_hedge_pool_lp_mint.supply == 0
@ErrorCode::ConstraintMintZeroSupply,
)]
pub vault_hedge_pool_lp_mint: Account<'info, Mint>;
```

Unchecked Oracle Account in CreateVaultMetadataState when used from an admin function

Finding ID: FYEO-AMULET-ID-03
Severity: **High**
Status: **Remediated**

Description

The `vault_oracle_account` is not checked for validity in the `CreateVaultMetadataState` function. This means that an inattentive or malicious admin could potentially pass in a malicious or incorrect oracle account when creating a vault.

Proof of Issue

File name: lib.rs
Line number: 181

```
/// CHECK: the switchboard oracle feed account
pub vault_oracle_account: AccountInfo<'info>,
```

File name: instructions/create_vault_metadata_state.rs
Line number: 53

```
self.vault_metadata_state.vault_oracle_account_key = self.vault_oracle_account.key();
```

Severity and Impact Summary

If a malicious or incorrect oracle account is used, it could lead to incorrect price data being used in the vault, which could have severe consequences for the vault's operation and the funds of its users.

Recommendation

Add checks to ensure that the `vault_oracle_account` is valid and comes from a trusted source. This may include checking that the account is owned by switchboard, and that it has the correct data format and structure. The account should be de-serialized as is done in `pub fn get_latest_trigger_status()` to ensure this won't fail later on where it would lead to a DoS situation.

ADMIN CAN DENY WITHDRAWAL OF FUNDS BY PAUSING VAULTS

Finding ID: FYEO-AMULET-ID-04
Severity: **Medium**
Status: **Remediated**

Description

The instructions `SettleVault`, `WithdrawRiskFund` and `WithdrawHedgeFund` can not be run if the admin pauses the vault by setting `vault_is_paused`. It appears that the vault is settled with an outcome in `AttemptToTrigger` and it would seem unnecessary for a vault to be paused afterwards.

Proof of Issue

File name: lib.rs
Line number: 438, 488, 568

```
constraint = !vault_metadata_state.vault_is_paused
@ErrorCode::ConstraintStateVaultIsPaused,
```

Severity and Impact Summary

A malicious admin could pause the vault in order to prevent users from withdrawing funds. The admin can then blackmail users in an attempt to force them to do as the admin demands.

Recommendation

Consider if the pause functionality is useful at this stage.

Hardcoded Program Initializer Authority

Finding ID: FYEO-AMULET-ID-05
Severity: **Low**
Status: **Remediated**

Description

The program initializer authority (PIA) is hardcoded in the source code, which makes it difficult to update or change the PIA in the future.

Proof of Issue

File name: lib.rs
Line number: 16

```
pub const PROGRAM_INITIALIZER_AUTH: Pubkey = Pubkey::new_from_array([ ... ]);
```

Severity and Impact Summary

If the hardcoded PIA is compromised, it would be difficult to replace it with a new one, which could result in a loss of control over the program.

Recommendation

Instead of hardcoding the PIA, consider using a configurable value that can be updated or changed if needed. This can be done by storing the PIA in an account or using a PDA (Program Derived Address) to manage the PIA.

LP MINTS CAN BE THE SAME ACCOUNT

Finding ID: FYEO-AMULET-ID-06
Severity: **Low**
Status: **Remediated**

Description

The two mint accounts `vault_risk_pool_lp_mint` and `vault_hedge_pool_lp_mint` can be the same account.

Proof of Issue

File name: lib.rs
Line number: 169

```
#[account(
    constraint = vault_risk_pool_lp_mint.freeze_authority.is_none()
@ErrorCode::ConstraintMintFreezeAuthNone,
    // owner = anchor_spl::token::ID @ErrorCode::OwnerTokenProgramID
)]
pub vault_risk_pool_lp_mint: Account<'info, Mint>,

#[account(
    constraint = vault_hedge_pool_lp_mint.freeze_authority.is_none()
@ErrorCode::ConstraintMintFreezeAuthNone,
    // owner = anchor_spl::token::ID @ErrorCode::OwnerTokenProgramID
)]
pub vault_hedge_pool_lp_mint: Account<'info, Mint>,
```

Severity and Impact Summary

In case these accounts are the same, the same token would be issued for both hedge and risk sides. Which would cause issues with settling and withdrawing.

Recommendation

Make sure that the mint accounts do not have the same key.

Lack of Input Validation in CreateVault

Finding ID: FYEO-AMULET-ID-07
Severity: **Low**
Status: **Remediated**

Description

The instruction `CreateVaultMetadataState` accepts several parameters, but there is no input validation for these parameters. This could lead to logical errors and unintended behavior if incorrect values are passed.

Proof of Issue

File name: instructions/create_vault_metadata_state.rs
Line number: 39

```
self.vault_metadata_state.vault_risk_pool.fee_pctg = risk_pool_fee_pctg;
...
self.vault_metadata_state.vault_hedge_pool.fee_pctg = hedge_pool_fee_pctg;

...
self.vault_metadata_state.vault_funding_stage_time = funding_stage_time;
self.vault_metadata_state.vault_locked_stage_time = locked_stage_time;
self.vault_metadata_state.vault_settlement_stage_time = settlement_stage_time;
self.vault_metadata_state.vault_withdrawal_waiting_time = withdrawal_waiting_time;
```

Severity and Impact Summary

An inattentive or malicious admin could potentially set times that do not make sense or set extreme fees as there is no limit on those, this could lead to unintended behavior and potential loss of user funds.

Recommendation

Add input validation and or range checks for all parameters passed to the `CreateVaultMetadataState` instruction. Ensure that all parameters are within the expected range and format before processing them.

NEW ADMIN IS NOT CO-SIGNER IN UPDATEADMIN

Finding ID: FYEO-AMULET-ID-08
Severity: **Low**
Status: **Remediated**

Description

The `AdminUpdateAdminAuthInfo` instruction allows the admin to update the admin auth key with limited checks on the new key. This could lead to a loss of control over the vault if the new key is unable to sign.

Proof of Issue

File name: lib.rs
Line number: 203

```
#[account(
    owner = system_program::ID @ErrorCode::OwnerSystemProgramID,
    constraint = new_vault_admin_auth.key() != vault_admin_auth.key()
@ErrorCode::ConstraintNonSameAdminKey,
    constraint = new_vault_admin_auth.lamports() >=
sysvar_rent.minimum_balance(new_vault_admin_auth.data_len())
@ErrorCode::ConstraintMinAccountRent,
)]
/// CHECK: pass in admin auth account
pub new_vault_admin_auth: AccountInfo<'info>,
```

Severity and Impact Summary

The admin could accidentally replace the admin auth key with an invalid key, leading to a loss of control over the vault.

Recommendation

Make `new_vault_admin_auth` a `Signer` that way it is guaranteed that someone has the keys for this account.

THE AUTHORITY ON THE LP MINT ACCOUNTS IS NOT CHECKED

Finding ID: FYEO-AMULET-ID-09
Severity: **Low**
Status: **Remediated**

Description

The mint authority of both the `vault_risk_pool_lp_mint` and `vault_hedge_pool_lp_mint` are not checked.

Proof of Issue

File name: lib.rs
Line number: 169

```
#[account(
    constraint = vault_risk_pool_lp_mint.freeze_authority.is_none()
@ErrorCode::ConstraintMintFreezeAuthNone,
    // owner = anchor_spl::token::ID @ErrorCode::OwnerTokenProgramID
)]
pub vault_risk_pool_lp_mint: Account<'info, Mint>,

#[account(
    constraint = vault_hedge_pool_lp_mint.freeze_authority.is_none()
@ErrorCode::ConstraintMintFreezeAuthNone,
    // owner = anchor_spl::token::ID @ErrorCode::OwnerTokenProgramID
)]
pub vault_hedge_pool_lp_mint: Account<'info, Mint>,
```

Severity and Impact Summary

Since the mint authority of these accounts is not checked, yet later assumed to be `vault_auth_pda` a vault can run into a DoS situation where nobody is able to deposit.

Recommendation

Verify that `vault_auth_pda` is the mint authority of those mint accounts.

THE VAULT POOL AND TREASURY TOKEN ACCOUNTS DO NOT CHECK THE DELEGATE

Finding ID: FYEO-AMULET-ID-10
Severity: **Low**
Status: **Remediated**

Description

There is no check for a `delegate` authority for the `vault_treasury_token_account` and `vault_pool_token_account` token accounts. The `close_authority` is not checked either.

Proof of Issue

File name: lib.rs
Line number: 157

```
#[account(
    constraint = *vault_admin_auth.key == vault_treasury_token_account.owner
@ErrorCode::ConstraintInvalidTAOwner,
    // owner = anchor_spl::token::ID @ErrorCode::OwnerTokenProgramID
)]
pub vault_treasury_token_account: Account<'info, TokenAccount>,

#[account(
    constraint = *vault_auth_pda.key == vault_pool_token_account.owner
@ErrorCode::ConstraintInvalidTAOwner,
    // owner = anchor_spl::token::ID @ErrorCode::OwnerTokenProgramID
)]
pub vault_pool_token_account: Account<'info, TokenAccount>,
```

Severity and Impact Summary

If a malicious admin adds a `delegate` to either token account, or does not properly check existing accounts, a delegated authority might steal tokens from these accounts. The `close_authority` may also be set on these accounts and should be checked as that allows closing of token accounts with zero balance or moving the balance of native token accounts to another account.

Recommendation

Add a check to verify that there is no delegate and close authority set on these accounts.

References

https://github.com/solana-labs/solana-program-library/blob/master/token/program/src/state.rs#L86

Unchecked token account balance for vault pool

Finding ID: FYEO-AMULET-ID-11
Severity: **Low**
Status: **Remediated**

Description

In the `create_vault_metadata_state` function, there is no check to ensure that the token account `vault_pool_token_account` has a zero balance. This may lead to unexpected behavior as the admin may initialize the vault with token accounts that have already been used.

Proof of Issue

File name: lib.rs
Line number: 163

```
#[account(
    constraint = *vault_auth_pda.key == vault_pool_token_account.owner
@ErrorCode::ConstraintInvalidTAOwner,
    // owner = anchor_spl::token::ID @ErrorCode::OwnerTokenProgramID
)]
pub vault_pool_token_account: Account<'info, TokenAccount>,
```

Severity and Impact Summary

The admin may initialize the vault with a token account that has already been used, leading to unexpected behavior and potential loss of funds.

Recommendation

Add constraints to check if the token account has a zero balance. Or make this account `init` and create it as a PDA.

Duplicate code execution in AttemptToTrigger

Finding ID: FYEO-AMULET-ID-12
Severity: **Informational**
Status: **Open**

Description

In `AttemptToTrigger` the `get_latest_trigger_status()` function is run twice. Once from the constraint macros and again in the `process()` function.

Proof of Issue

File name: lib.rs
Line number: 411

```
constraint =
vault_metadata_state.get_latest_trigger_status(sysvar_clock.unix_timestamp as u64,
&vault_token_mint, &vault_oracle_account).2 != VaultTriggeredResultNone as u64
@ErrorCode::ConstraintEmptyNewTriggeredResult,
```

File name: instructions/attempt_to_trigger.rs
Line number: 7

```
pub fn process(&mut self) -> Result<()> {
    let oracle_feed_account = &self.vault_oracle_account;
    let (raw_oracle_price, oracle_price, latest_status) =
self.vault_metadata_state.get_latest_trigger_status(self.sysvar_clock.unix_timestamp
as u64, &self.vault_token_mint, oracle_feed_account);
```

Severity and Impact Summary

There is no security impact, it is an optimization to simplify the code.

Recommendation

The `process()` function could return the same error in case the result is `VaultTriggeredResultNone`.

In the CreateVault instruction the vault admin should co-sign

Finding ID: FYEO-AMULET-ID-13
Severity: **Informational**
Status: **Remediated**

Description

The admin for this vault is not a co-signer of the instruction.

Proof of Issue

File name: lib.rs
Line number: 138

```
#[account(
    owner = system_program::ID @ErrorCode::OwnerSystemProgramID,
)]
/// CHECK: pass in admin auth account
pub vault_admin_auth: AccountInfo<'info>,
```

Severity and Impact Summary

It is possible to supply an account that nobody has the keys for making the use of admin functionality impossible.

Recommendation

Consider making the `vault_admin_auth` account a `Signer` for this instruction.

MATH INCONSISTENCY

Finding ID: FYEO-AMULET-ID-14
Severity: **Informational**
Status: **Open**

Description

Checked math is inconsistently used in the code base. The cargo setting `overflow-checks=true` is used which would cause a panic in case of math overflows but most of the code explicitly uses checked math anyway. There is however one exception.

Proof of Issue

File name: states/metadata_state.rs
Line number: 139

```
let oracle_price = (10.0_f64.powi(vault_token_mint.decimals as
i32).mul(raw_oracle_price)) as u64;
```

Severity and Impact Summary

This is not a security issue as the whole code base is covered against overflows using `overflow-checks=true`. It is just a suggestion to improve maintainability and prevent any issues in the future should the compiler flags be changed.

Recommendation

Consider using explicit checked math throughout the code base.

THERE IS NO WAY TO CLEAN UP OLD ACCOUNTS

Finding ID: FYEO-AMULET-ID-15
Severity: **Informational**
Status: **Open**

Description

There is currently no way to cleanup old accounts. Solana accounts have a rent exemption fee that can be refunded in case an account is closed. Since each vault has a number of accounts associated with it, those could potentially be closed once the vault is settled and users have withdrawn all their funds. Mint accounts can also be closed in the `Token2022` standard which could be used for the LP mints. A compressed version of the vault could also be used for archival purposes.

Proof of Issue

It is an absence of code.

Severity and Impact Summary

No security impact. This would only optimize storage and make it possible to regain smaller amounts from rent exemption fees.

Recommendation

Consider adding such functionality.

References

https://spl.solana.com/token-2022/extensions#example-closing-a-mint

UNNECESSARY CONSTRAINTS

Finding ID: FYEO-AMULET-ID-16
Severity: **Informational**
Status: **Open**

Description

There are some unnecessary checks in the code that could be removed.

Proof of Issue

File name: lib.rs
Line number: various

```
constraint = staker_token_account.to_account_info().lamports() >=
sysvar_rent.minimum_balance(spl_token::state::Account::LEN)
@ErrorCode::ConstraintMinTokenAccountRent,
```

The check is redundant since the token accounts are already required to be rent-exempt. This check is done in `DepositRiskFund`, `DepositHedgeFund`, `WithdrawRiskFund`, and `WithdrawHedgeFund`.

File name: lib.rs
Line number: various

```
#[account(
    address = sysvar::clock::ID @ErrorCode::AddressInvalidSysvarClock
)]
pub sysvar_clock: Sysvar<'info, Clock>,

#[account(
    address = system_program::ID @ErrorCode::AddressInvalidSPID
)]
pub system_program: Program<'info, System>,

#[account(
    address = anchor_spl::token::ID @ErrorCode::AddressInvalidTPID
)]
pub token_program: Program<'info, Token>,

#[account(
    address = sysvar::rent::ID @ErrorCode::AddressInvalidSysvarRent
)]
pub sysvar_rent: Sysvar<'info, Rent>,
```

Anchor does check the addresses of specific account types, such as `System`, `Clock`, `Token` and `Rent`.

Severity and Impact Summary

No security impact. This would be a code optimization which could increase the readability.

Recommendation

Remove duplicate constraints from the affected functions.

WITHDRAW FUNCTIONS DO NOT CHECK TOKEN AMOUNT AGAINST WITHDRAW AMOUNT

Finding ID: FYEO-AMULET-ID-17
Severity: **Informational**
Status: **Open**

Description

The `WithdrawRiskFund` and `WithdrawHedgeFund` functions check that `lp_token_account.amount > 0`. The amount of tokens the user is attempting to withdraw is known as `lp_amount`. The check could instead make sure that the token account contains at least `lp_amount` worth of tokens.

Proof of Issue

File name: lib.rs
Line number: 533

```
constraint = staker_lp_token_account.amount > 0 @ErrorCode::ConstraintNonZeroLPAmount,
```

File name: lib.rs
Line number: 613

```
constraint = hedger_lp_token_account.amount > 0 @ErrorCode::ConstraintNonZeroLPAmount,
```

The amount of tokens to be withdrawn is known:

```
#[instruction(lp_amount: u64)]
```

Severity and Impact Summary

No security impact. Token transfers would not work without the user having the required amount of tokens. Any amount check here is purely for custom errors.

Recommendation

Consider checking against the actual withdraw amount.

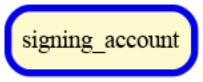
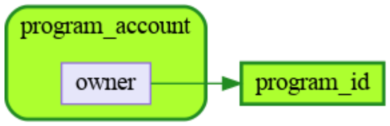
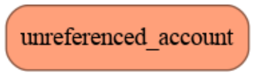
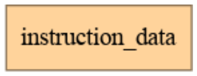
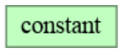
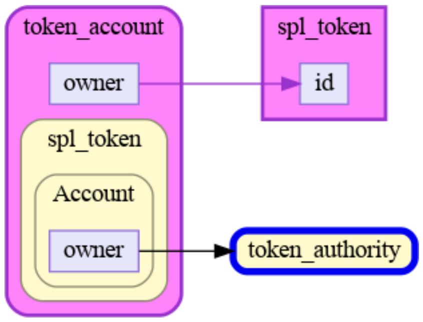
Relationship Graphs

A relationship graph shows relational requirements to the input of an instruction. Notice, that it does not show outcome of the instruction.

Relationship graphs are used to analyze insufficient authorization and unchecked account relations. Insufficient authorization allows unintended access. Unchecked account relations may allow account and data injection resulting in unintended behavior and access.

Various styles are used to highlight special properties. Accounts are shown as boxes with round corners. An account box may contain smaller boxes indicating relevant account data.

The following table shows the basic styles for the relationship graphs.

	Round boxes with thick blue borders indicate accounts required to sign the transaction.
	Green round boxes highlight accounts required to be owned by the program itself.
	Red round boxes indicate accounts where ownership is not validated. Further analysis should be made to ensure this does not allow account injection attacks.
	Orange boxes indicate instruction data. As instruction data does not originate from the blockchain, it may open for data injection attacks. Caution must be taken if used for account validation.
	Green boxes indicate constants. It may refer to either hardcoded values or library code.
 token_account owner spl_token Account owner spl_token id token_authority	Inner boxes are used to indicate data structures to ease readability. Additional colors may also be used to highlight account ownership from different programs.

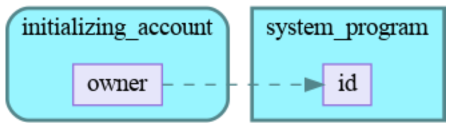
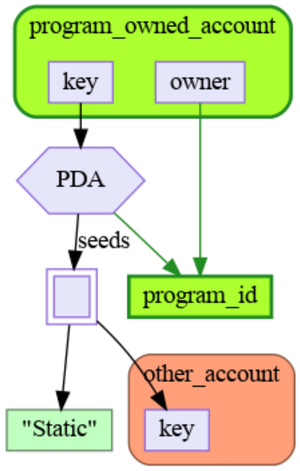
	Dashed lines indicate implicitly required relations. For example, relations required by another program during a cross program invocation. This is emphasized as the program cannot guarantee the behavior of external programs.
 program_owned_account key owner PDA seeds program_id other_account key "Static"	Program derived addresses are calculated using the `find_program_address` or `create_program_address` functions. To illustrate the relations to the input used for the PDA calculation, a diamond shaped box is used to show the PDA and a double-edged box to gather the seeds. As seed injection may lead to account injection, it is important that all input to the PDA is verified.

Table 3: Legend for relationship graphs

CreateVaultMetadataState

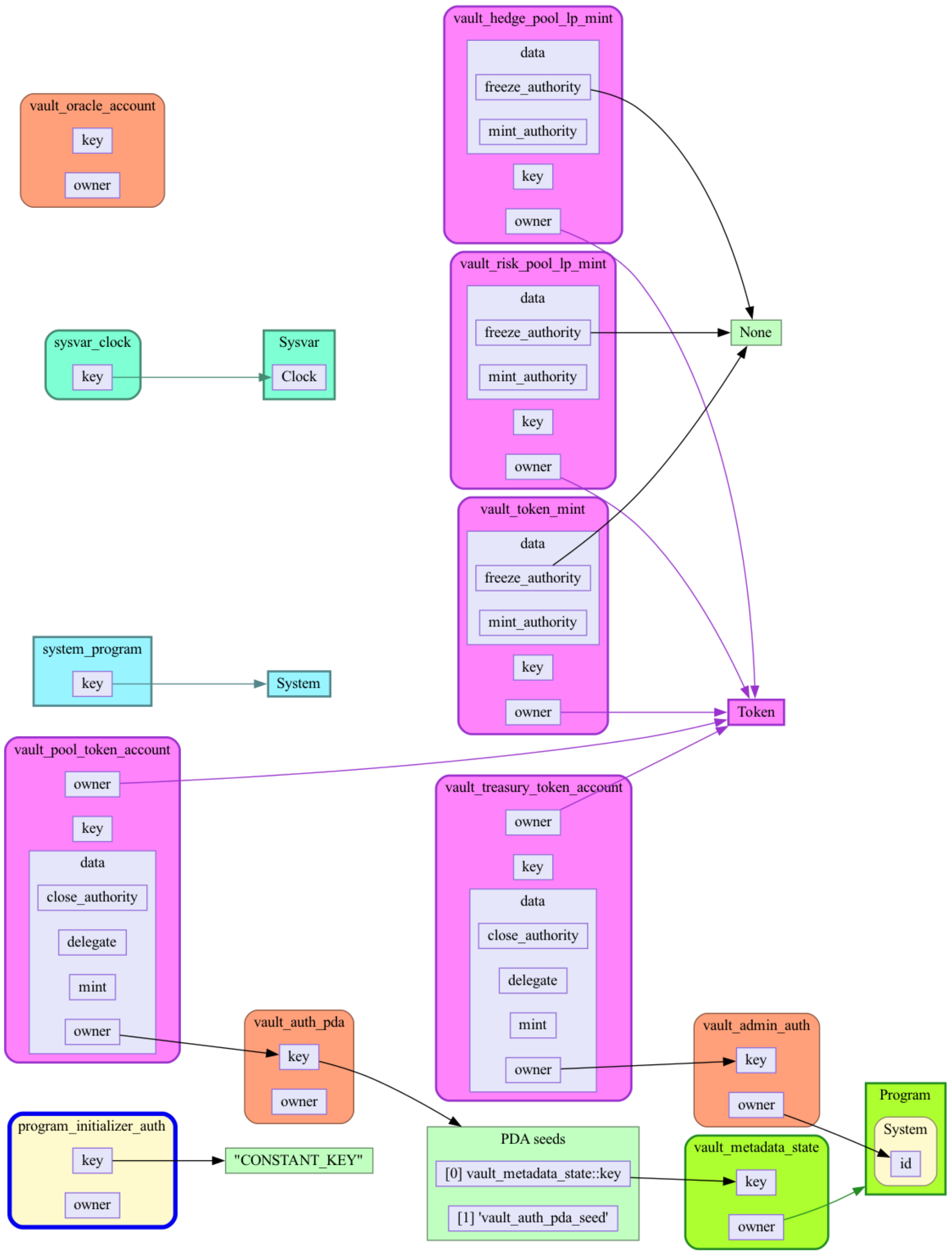


Figure 2: CreateVaultMetadataState graph

There are no checks for the `vaule_oracle_account` which could lead to DoS situations when the vault tries to settle. Neither the `vault_treasury_token_account` nor `vault_pool_token_account` are checked for a delegate or close authority which could enable someone to transfer funds from these accounts. The `vault_admin_auth` account is not a signer even though this account serves as an admin. The two mint accounts could be the same account as neither key is checked. Also the LP mint accounts do not check who the mint authority is which is later assumed to be `vault_auth_pda` but this might lead to a situation where nobody can deposit as no LP tokens can be minted. Neither the `vault_treasury_token_account` nor `vault_pool_token_account` mints are checked which could lead to user funds being stuck as attempting to pay the fee to the treasury will not work.

AdminUpdateAdminAuthInfo

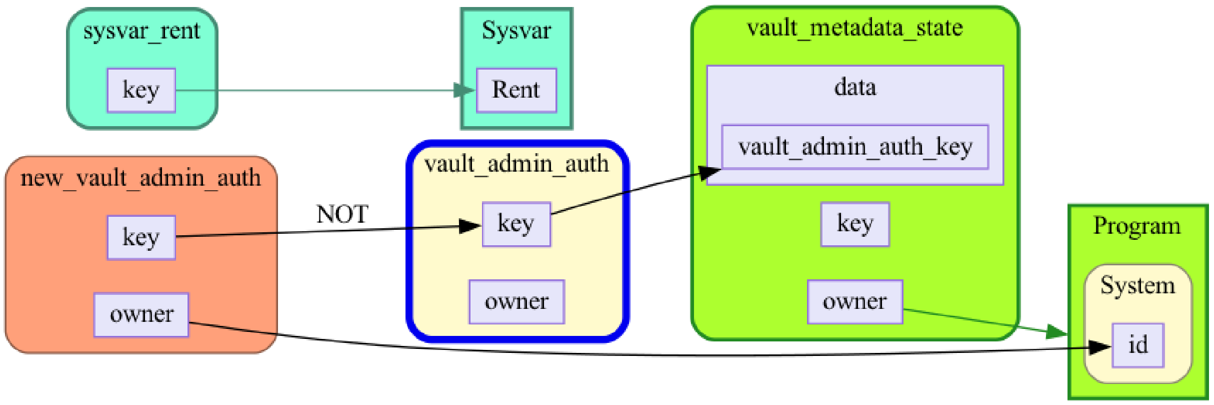


Figure 3: AdminUpdateAdminAuthInfo graph

The `new_vault_admin_auth` is not a signer of the transaction.

AdminUpdatePauseInfo

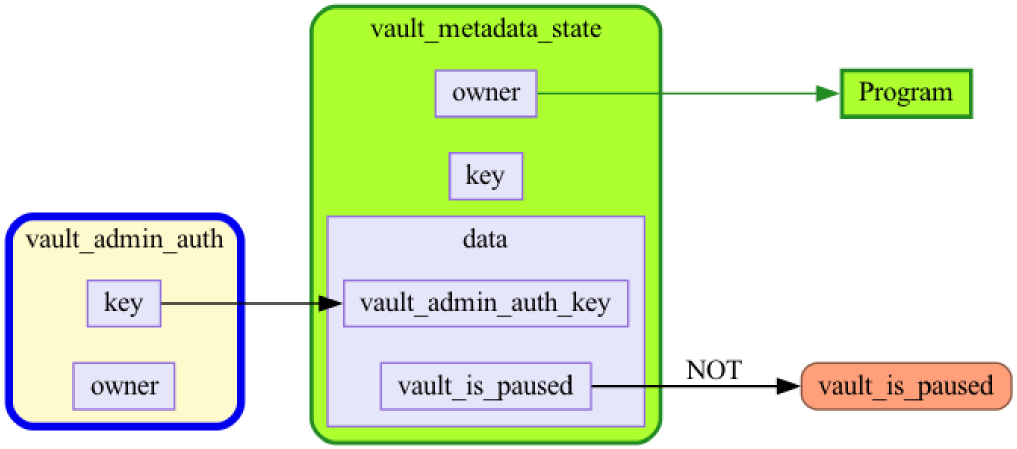


Figure 4: AdminUpdatePauseInfo graph

Allows pausing the vault at any stage.

DepositRiskFund

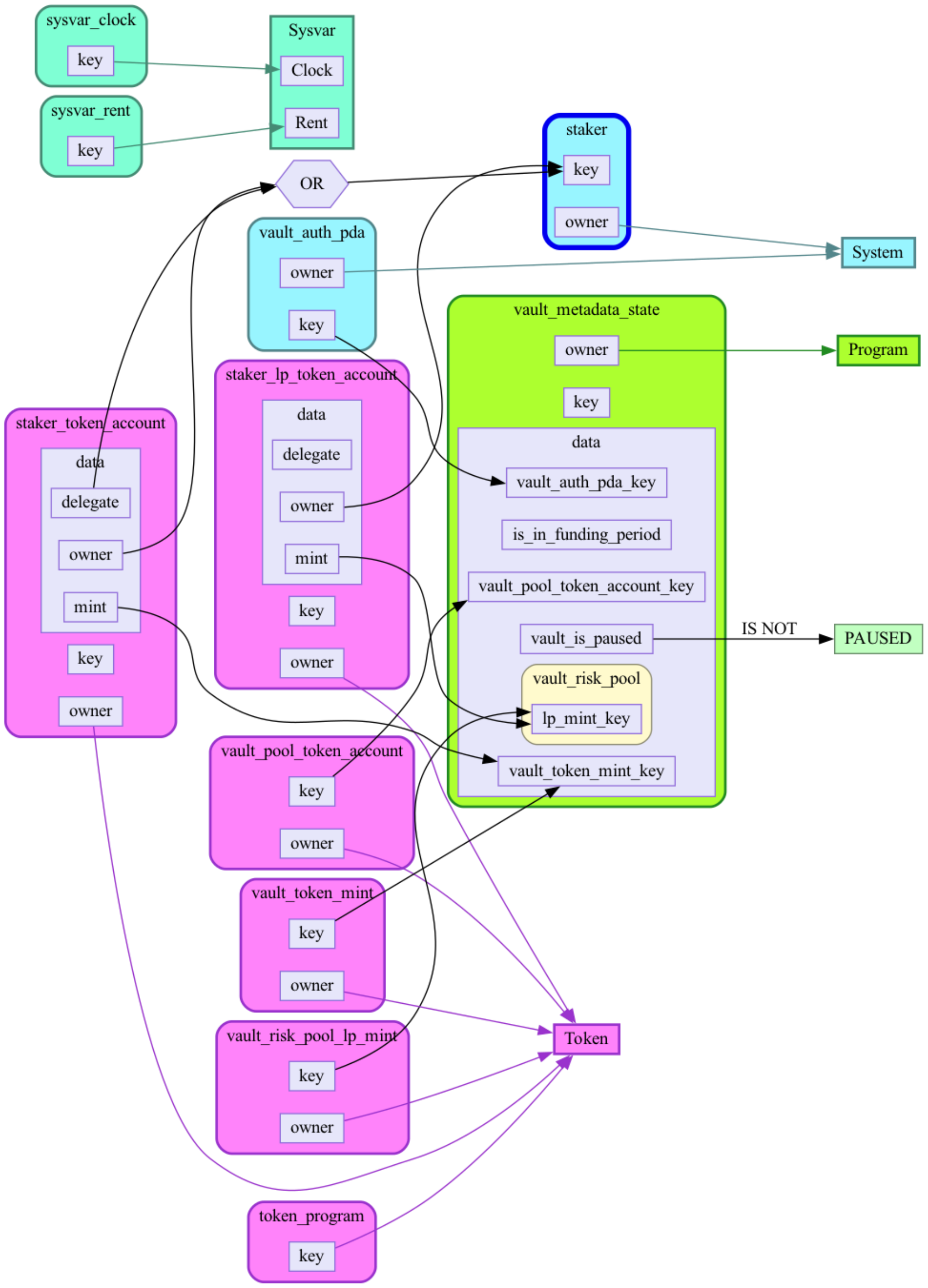


Figure 5: DepositRiskFund graph

Allows depositing tokens into a non paused vault during the funding period in return for LP tokens.

DepositHedgeFund

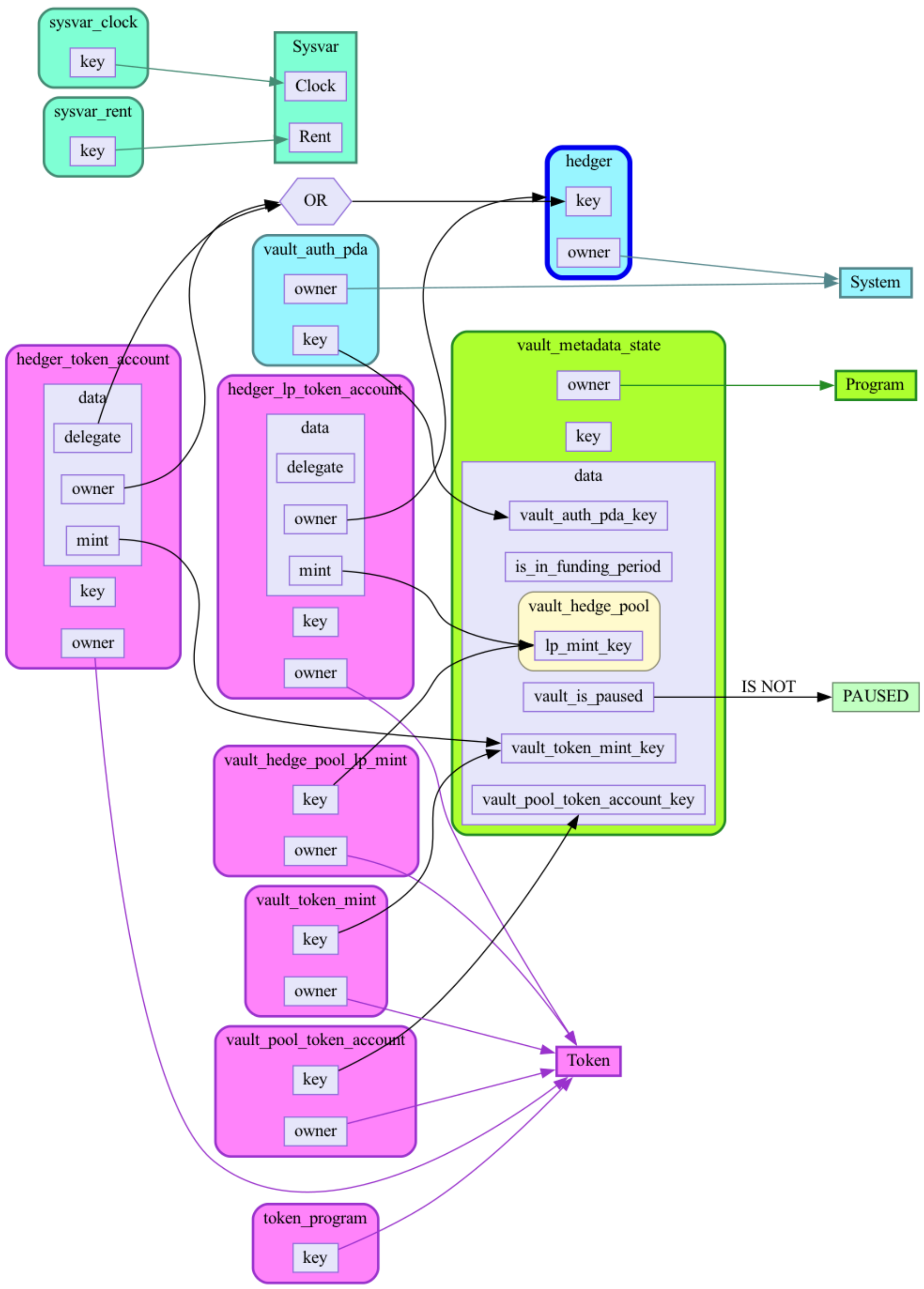


Figure 6: DepositRiskFund graph

Allows depositing tokens into a non paused vault during the funding period in return for LP tokens.

AttemptToTrigger

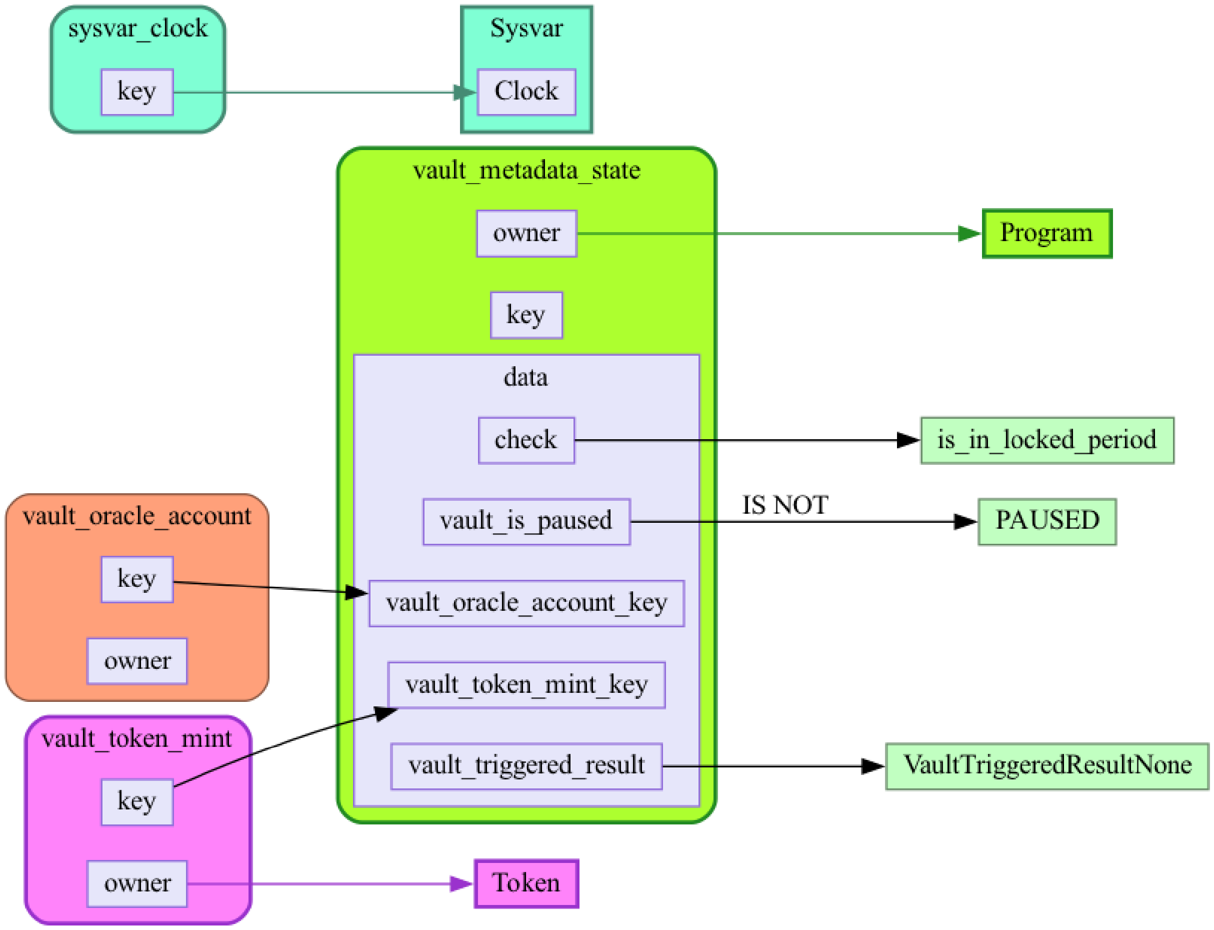


Figure 7: AttemptToTrigger graph

An un-permissioned function to trigger a non-paused vault in the locked period. No previous trigger result must be set.

SettleVault

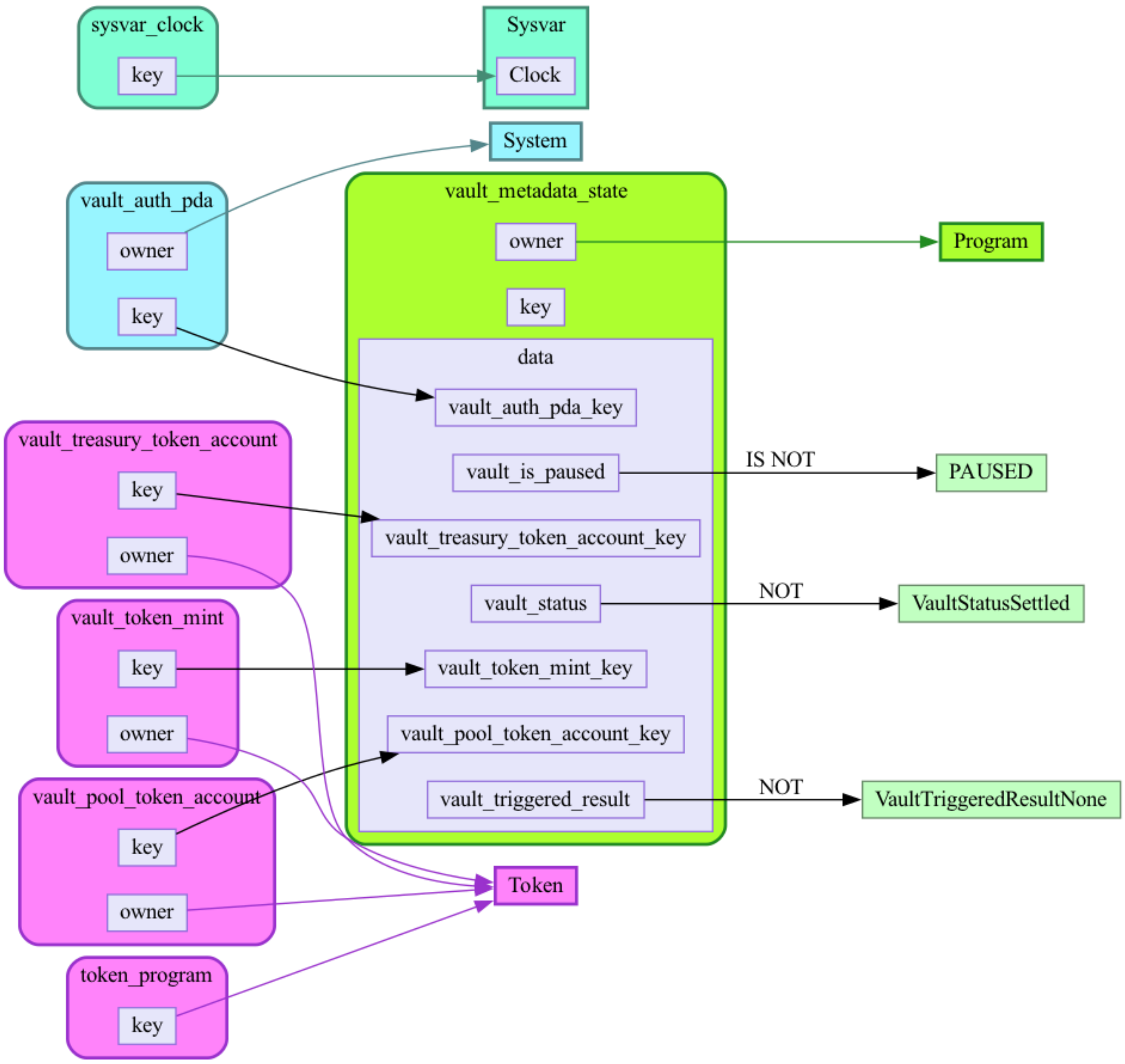


Figure 8: SettleVault graph

Settle a non paused vault provided conditions are met.

WithdrawRiskFund

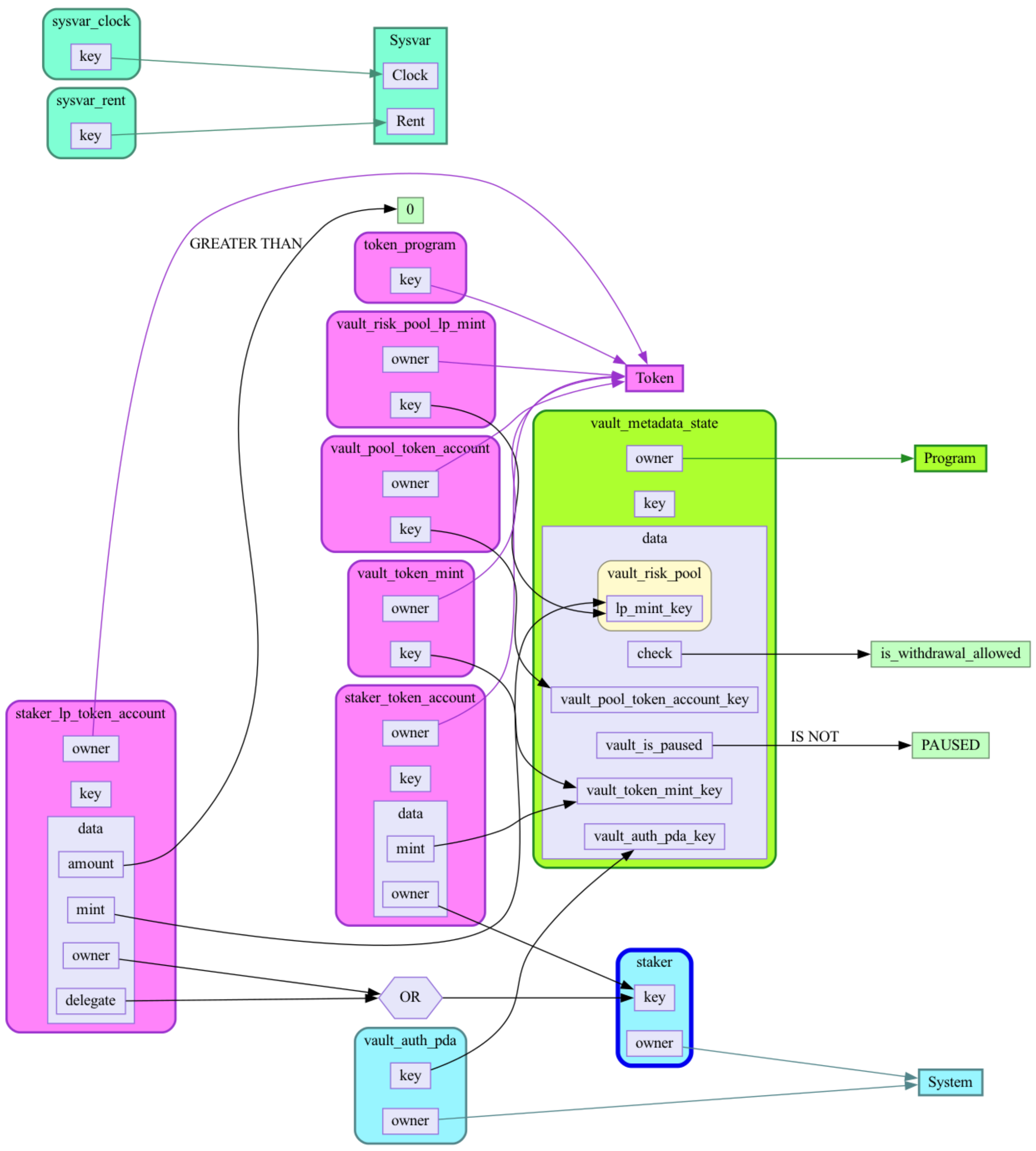


Figure 9: WithdrawRiskFund graph

If the vault isn't paused and withdrawal is allowed, burns LP tokens from the user in exchange for underlying vault tokens.

WithdrawHedgeFund

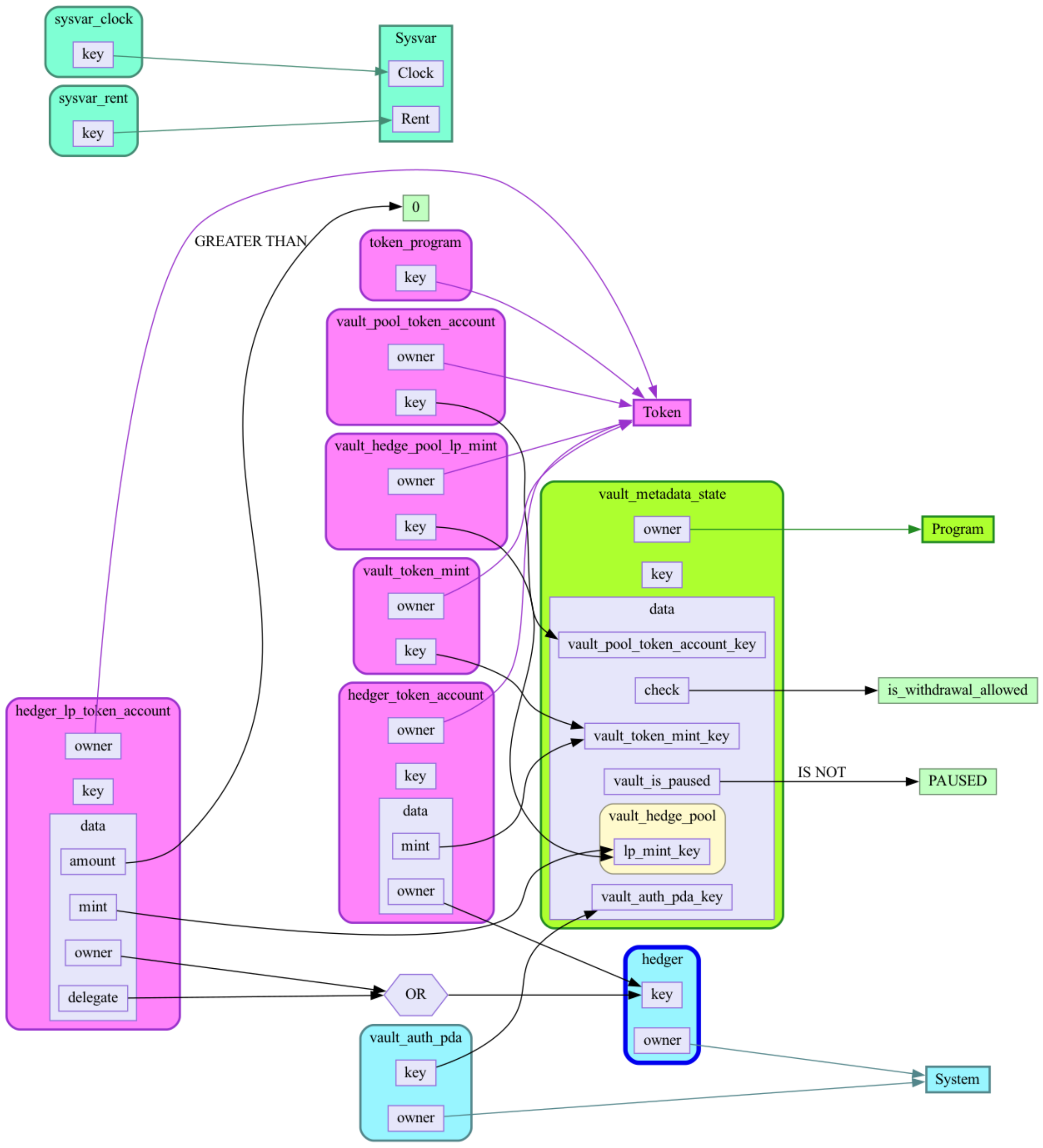


Figure 10: WithdrawHedgeFund graph

If the vault isn't paused and withdrawal is allowed, burns LP tokens from the user in exchange for underlying vault tokens.

OUR PROCESS

METHODOLOGY

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

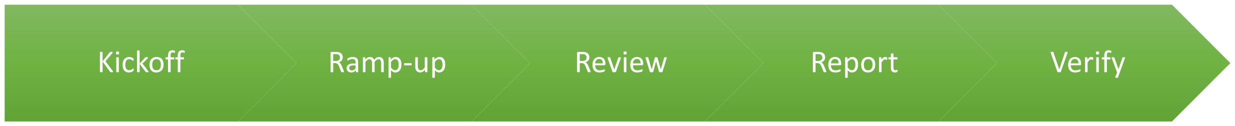


Figure 11: Methodology Flow

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