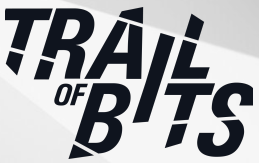

FIVA Yield Tokenization Protocol

Security Assessment

May 30, 2025

Prepared for:
Andrei Yazepchyk
FIVA

Prepared by: **Tarun Bansal, Nicolas Donboly, Coriolan Pinhas, Quan Nguyen**

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Mary O'Brien, Project Manager
mary.obrien@trailofbits.com

The following engineering director was associated with this project:

Jim Miller, Engineering Director, Cryptography
james.miller@trailofbits.com

The following consultants were associated with this project:

Tarun Bansal, Consultant
tarun.bansal@trailofbits.com

Nicolas Donboly, Consultant
nicolas.donboly@trailofbits.com

Coriolan Pinhas, Consultant
coriolan.pinhas@trailofbits.com

Quan Nguyen, Consultant
quan.nguyen@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
February 27, 2025	Pre-project kickoff call
March 10, 2025	Status update meeting #1
March 14, 2025	Status update meeting #2
March 28, 2025	Status update meeting #3
April 04, 2025	Status update meeting #4
April 11, 2025	Status update meeting #5
April 16, 2025	Status update meeting #6
April 25, 2025	Delivery of report draft
April 25, 2025	Report readout meeting
May 30, 2025	Delivery of final comprehensive report

Executive Summary

Engagement Overview

FIVA engaged Trail of Bits to review the security of the yield tokenization protocol. This protocol allows users to break their yield-earning tokens into principal and yield tokens to provide fixed-yield assets and high-leverage investments. The protocol also implements an AMM to allow users to swap the yield-earning tokens with the principal and yield tokens.

A team of one consultant conducted the review from March 3 to April 25, 2025, for a total of six engineer-weeks of effort. Our testing efforts focused on analyzing the access control system, input data validation, and error handling flow, identifying race conditions among user actions, corruption of contract state, arithmetic operation precision loss, and vulnerabilities to denial-of-service attacks. With full access to source code and documentation, we performed static and dynamic testing of the FIVA protocol smart contracts, using automated and manual processes. We did not review the deployment scripts and off-chain components during this engagement.

Observations and Impact

The codebase is structured well and broken down into small smart contracts that handle limited functionality to manage complexity. The documentation and inline code comments help developers and reviewers navigate the code and follow user action message flows through different smart contracts.

We discovered four high-severity issues arising from insufficient or incorrect data validation checks (TOB-FIVA-2, TOB-FIVA-8) and incorrect use of an intermediate contract to store temporary state (TOB-FIVA-3, TOB-FIVA-9). We identified race conditions that could lead to unexpected protocol behavior (TOB-FIVA-13). We also found low-severity and informational issues related to input data validation, incorrect gas checks, use of incorrect hard-coded values, arithmetic operation integer overflow, and inefficient access control checks.

Additionally, administrative functions are implemented to update every storage value of the smart contract without validation checks. This introduces issues related to incorrect configuration (TOB-FIVA-4) and inconsistent system state. A human mistake in any administrative operation can break the protocol and lock user funds.

Recommendations

Based on the codebase maturity evaluation findings identified during the security review, Trail of Bits recommends that FIVA take the following steps before deploying the protocol smart contracts in production:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or any refactoring that may occur when addressing other recommendations.

- **Analyze and document the effect of race conditions.** Document in diagrams all the different paths of user action flow, including the number of transactions and their state changes. Analyze these flow diagrams to detect race conditions and determine their effects' scope, and document them to discover and resolve security issues.
- **Remove unnecessary administrative functions.** Review all of the administrative functions to analyze if they are required for the core functionality of the protocol; if not, remove them. Group configuration update function to keep configuration values in sync in a single contract or multiple contracts.
- **Improve the test suite.** Consider invalid or malicious user inputs to test protocol stability against them. Add test cases for getter functions and cover more code paths, such as the locked token amount in the `Pool` contract.

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	5
Medium	1
Low	2
Informational	6
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Configuration	2
Data Validation	6
Denial of Service	1
Testing	1
Timing	1
Undefined Behavior	3

Project Goals

The engagement was scoped to provide a security assessment of the FIVA yield tokenization protocol. Specifically, we sought to answer the following non-exhaustive list of questions:

- Are gas checks effective?
- Can access control checks be bypassed?
- Are user inputs validated sufficiently?
- Can users withdraw more than their share?
- Can an attacker steal funds from other users?
- Can the system reach an inconsistent state?
- Can an attacker make any of the smart contracts unusable?
- Can administrator actions break the smart contracts?
- Can an attacker grieve other users by blocking their operations?
- Can race conditions lead to unexpected outcomes?
- Are the mathematical formulas implemented correctly?
- Can an attacker exploit precision loss?

Project Targets

The engagement involved reviewing and testing the following target.

contracts_v2

Repository	https://github.com/Fiva-protocol/contracts_v2
Version	`commit e74ada0`
Type	FunC
Platform	TVM

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- **General:** We reviewed the whole codebase for common FunC and TON blockchain flaws, such as missing or incorrect gas checks, issues with bounced message handling, and message parsing and building errors. We checked the use of modifying functions, message and storage cell limits, error handling logic, maintenance of the contracts' TON balance, and contract deployment and initialization, all for correctness.

- **`SYMinter` contract:** The `SYMinter` contract is used to wrap all of the yield-bearing tokens in the SY Jetton, which allows the FIVA protocol to interact with all the yield-bearing tokens with a standard interface of the SY Jetton. We checked whether user actions are completed and errors are handled correctly without leaving the system in an inconsistent state. We checked whether users can lose their assets and whether attackers can steal user funds. We also looked for vulnerabilities of denial-of-service attacks on the SY Jetton smart contracts.

- **`YTMinter` contract:** The `YTMinter` contract breaks the SY tokens into PT and YT tokens for yield tokenization. The PT token represents the principal part, and the YT token represents the yield part of the underlying yield-bearing token. In addition to the checks done on the `SYMinter` contract, we looked for issues related to the use of the `RedeemDeposit` contract, focusing on the access control checks and race conditions.

- **`Pool` contract:** The `Pool` contract implements a cube stable AMM for the SY, PT, and YT tokens. We checked the cube stable mathematical functions for the correct implementation of the AMM formulas. We looked for issues related to the index value sync between the `Pool` and the `YTMinter` contract. We checked if users can manipulate the pool reserves to steal funds from other users. We also reviewed the `Pool`, `YTMinter`, and `SYMinter` contract integration for chained operations to check if these operations are completed and do not leave the system in an inconsistent state. We also looked at race conditions in these chained operations to find issues leading to loss of funds or denial-of-service vulnerabilities.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. During this project, we were unable to perform comprehensive testing of the following system elements, which may warrant further review:

- Prophet smart contract

- NFT collection and item smart contracts
- Curve stable and constant production AMM markets mathematical functions
- PT to YT and YT to PT swap operation handler functions
- Stake Jetton smart contracts
- Unlimited Jetton smart contracts
- Off-chain components
- Deployment scripts

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The protocol uses simple arithmetic operations to compute the YT token interest, swap, and liquidity token amounts for the AMM. The mathematical formulas are well documented. We identified an integer overflow issue in a calculation performed by the protocol, as described in TOB-FIVA-10. Rounding directions are not explicit; operations always round in the default direction. Performing an integer overflow and precision loss analysis to determine an effective precision scale for storing amounts would help mitigate the system's integer overflow and loss-of-precision risks.	**Moderate**
Auditing	The FIVA team has deployed an off-chain monitoring system to monitor smart contract behavior and transaction data. The current system is focused on monetary transactions and data analytics. Adding new features to the monitoring system to detect an inconsistent system state, an unauthorized transaction, or an unexpected state change will help quickly detect and respond to a security incident.	**Moderate**
Authentication / Access Controls	The protocol implements an effective access control system to prevent unauthorized access. All privileged functions have some form of access control following the principle of least privilege. The test suite includes test cases for positive and negative access control checks. However, we found multiple access control issues resulting from insufficient checks on the message sender (TOB-FIVA-3, TOB-FIVA-9). Documenting the authentication checks for all user actions and adding test cases for these checks can help prevent such issues. The system can also benefit from additional documentation	**Moderate**

	specifying the roles, their privileges, and the process of granting and revoking them.	
Complexity Management	The codebase is generally well structured and broken down into small contracts and functions. Each function has a specific and clear purpose and includes inline documentation. Core functions are straightforward to test via unit tests or automated testing.	**Satisfactory**
Decentralization	In the current state, privileged actors have complete control over the protocol parameters, configuration, and upgradeable contracts. Issues TOB-FIVA-2 and TOB-FIVA-14 show that privileged actors can use the protocol to their benefit and profit from it. However, the FIVA team stated that initially, these privileged roles will be assigned to multisignature wallets with a timelock functionality. Eventually, the contracts will be immutable, and the ownership of contracts will be dropped, making the protocol decentralized.	**Weak**
Documentation	The documentation provided consists of a high-level description of all user actions with message flow diagrams. Mathematical formulas are not documented; a formula simulation is provided to test correctness. The codebase includes extensive inline documentation, making it easy to understand the code. The documentation can be expanded to include privilege roles, their responsibilities, and the process to execute all maintenance tasks.	**Satisfactory**
Low-Level Manipulation	There are no instances of Fift assembly code in the protocol files.	**Not Applicable**
Testing and Verification	The test cases do not cover the whole protocol codebase or features, and the test coverage is not 100% for several test files. Considering the TOB-FIVA-14 issue, we recommend improving the test suite by adding adversarial cases such as invalid configurations and incorrect action parameters. Additionally, the tests should use values that closely match those that will be used in production. For	**Weak**

	instance, there are no test cases with non-zero values for locked PT or SY tokens in the AMM Pool. Adding test cases for the getter function can help detect issues such as TOB-FIVA-11.	
Transaction Ordering	Even though the TON blockchain network does not allow users to time when transactions are executed, race conditions among user actions can affect the protocol's stability. The FIVA team minimized smart contract state synchronization risks by having the protocol collect all the required data and enforcing all system checks in one place. However, the risks of race conditions are not clearly identified or documented. We identified race conditions that could enable timing attacks, as described in issue TOB-FIVA-13. Creating a detailed user action flow diagram with the number of transactions and state changes from every transaction can help discover race conditions.	**Moderate**

Summary of Findings

The table below summarizes the findings of the review, including details on type and severity.

ID	Title	Type	Severity
1	Lack of a two-step process for critical operations	Data Validation	Low
2	Lack of validation checks in the upgrade_storage operation handler	Data Validation	High
3	An attacker can prevent the redemption of YT and PT tokens from the YTMinter contract	Denial of Service	High
4	Users can lose funds because of incorrect SY token configurations in the YTMinter contract	Configuration	Medium
5	The SYWallet contract is not tested	Testing	Informational
6	Lack of a gas check in the wrap operation handler	Data Validation	Informational
7	Curve stable swap AMM is not usable	Configuration	Informational
8	An incorrect balance check for the PT-to-SY swap can lead to a loss of funds	Undefined Behavior	High
9	An attacker can grieve users by completing their liquidity provision operation	Undefined Behavior	High
10	An integer overflow in the cube stable market invariant calculation can make the AMM unusable for swaps	Data Validation	Informational
11	The YTMinter contract's get_claimable_interest function deducts the protocol fee twice	Data Validation	Informational

ID	Title	Type	Severity
1	Lack of a two-step process for critical operations	Data Validation	Low
2	Lack of validation checks in the upgrade_storage operation handler	Data Validation	High
3	An attacker can prevent the redemption of YT and PT tokens from the YTMinter contract	Denial of Service	High
12	Incorrect forward value when minting PT in function mint_py_jettons	Undefined Behavior	Low
13	Race condition in YT swap and index update can lead to loss of funds	Timing	High
14	Lack of validation checks in admin action handlers	Data Validation	Informational

Detailed Findings

1. Lack of a two-step process for critical operations	
Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-FIVA-1
Target: `contracts/SY/generic/maintance.fc`	

Description

When called, the `change_admin` operation handler function immediately sets the admin of the `SYMinter` contract. Using a single step to make such a critical change is error-prone; if the function is called with erroneous input, the results will be irrevocable.

```
if (op == op::change_admin) {
    ;; change the admin of the contract
    check_sender_is_admin(sender_address);
    storage::admin_address = in_msg_body~load_msg_addr();
    save_data();
    throw(successExitCode);
}
```

Figure 1.1: The `change_admin` operation handler of the `SYMinter` contract
`contracts/SY/generic/maintance.fc#L9-L15`

The same issue also affects the `change_admin` operation handler of the `EvaaSyMinter`, `YTMinter`, `PTMinter`, `UnlimitedMinter`, and `UnlimitedMinterV2` contracts and the `change_owner` operation handler of the `Pool` contract.

Exploit Scenario

Alice executes the `change_admin` operation to change the contract administrator but accidentally enters the wrong address. She permanently loses access to the contract.

Recommendations

Short term, use two steps for transferring critical roles: the first step to propose a new account for the role, and the second step to accept the role from the new account itself.

Long term, identify and document all possible actions that privileged accounts can take, along with their associated risks. This will facilitate codebase reviews and prevent future mistakes.

2. Lack of validation checks in the upgrade_storage operation handler	
Severity: **High**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-FIVA-2
Target: `contracts/SY/generic/maintance.fc`	

Description

The `upgrade_storage` operation saves the user-provided cell as contract storage data without validating its data. A malformed data cell can make the contract unusable.

The `upgrade_storage` operation handler of the `SYMinter` contract does not validate that the user-provided data cell includes all of the required storage fields in the correct format. A malformed, incomplete, or empty cell can be saved as the contract storage data, causing an exception from the `load_data` function of the contract. This will make the contract unusable, as all the user actions will throw an exception.

```
if (op == op::upgrade_storage) {
    check_sender_is_admin(sender_address);
    cell new_storage = in_msg_body~load_ref();
    set_data(new_storage);
    throw(successExitCode);
}
```

Figure 2.1: The `upgrade_storage` operation handler of the `SYMinter` contract
`contracts/SY/generic/maintance.fc#L97-L102`

The same issue also affects the `upgrade_storage` operation handler of the `EvaaSyMinter`, `YTMinter`, `PTMinter`, `UnlimitedMinter`, `UnlimitedMinterV2`, `Collection`, and `Pool` contracts.

Exploit Scenario

Alice upgrades the `SYMinter` contract storage but accidentally enters an incomplete data cell and saves it as the contract storage data. After this, every user action fails because the `load_data` function cannot parse the required storage data from the stored incomplete data cell and throws an exception.

Recommendations

Short term, implement a validation check in the `upgrade_storage` operation handler to ensure that the new data cell includes all the storage data fields in the correct format and can be parsed without an exception.

Long term, identify and document all possible actions that privileged accounts can take, along with their associated risks. This will facilitate codebase reviews and prevent future mistakes.

3. An attacker can prevent the redemption of YT and PT tokens from the YTMinter contract

Severity: **High**	Difficulty: **Medium**
Type: Denial of Service	Finding ID: TOB-FIVA-3
Target: `contracts/YT/redeem.fc`	

Description

An attacker can complete a user's redemption of YT and PT tokens from the `YTMinter` contract by sending the `redeem` message with the user as the receiver. This allows attackers to prevent users from redeeming their YT and PT tokens for SY tokens.

The FIVA protocol allows users to redeem their YT and PT tokens for SY tokens before maturity. This redemption process requires users to transfer both YT and PT tokens to the `YTMinter` contract with a `forward_payload`, including a `redeem` or `reddem_and_unwrap` operation, and the recipient of the SY tokens. On the first token transfer, the `YTMinter` contract deploys a `RedeemDeposit` contract with the `register_jetton` message. The `RedeemDeposit` contract stores the `YTMinter` address, the recipient address, and the PT or YT token amount transferred by the user:

```
() register_jetton(slice recipient_addr, cell redeem_deposit_code, int pt_amount,
int yt_amount, int query_id, int unwrap) impure {
    cell state_init = calculate_redeem_deposit_state_init(my_address(),
recipient_addr, redeem_deposit_code);
    slice redeem_deposit_addr = calc_redeem_deposit_address(state_init);
    var msg_body = begin_cell()
        .store_op(op::register_jetton)
        .store_query_id(query_id)
        .store_coins(pt_amount)
        .store_coins(yt_amount)
        .store_uint(unwrap, 1)
        .end_cell();
    var msg = begin_cell()
        .store_msg_flag(msg_flag::bounceable)
        .store_slice(redeem_deposit_addr)
        .store_coins(0)
        .store_msgbody_prefix_stateinit(state_init, msg_body)
        .end_cell();
    send_raw_message(msg, CARRY_REMAINING_BALANCE);
}
```

Figure 3.1: The `register_jetton` function of the `YTMinter` contract `contracts/YT/redeem.fc#L20-L37`

On the second token transfer, the `YTMinter` computes a `RedeemDeposit` address using self-address and the provided recipient address as the `init_state` data and sends the `register_jettons` message to it. The same recipient as the first transfer will result in the same `RedeemDeposit` address computation for the second transfer because of the same `init_state` data. When the user has transferred a non-zero amount of both the PT and YT tokens, The `RedeemDeposit` contract sends a `finalize_redeem` message to the `YTMinter` contract and destroys itself:

```
if (op == op::register_jetton) {
    int pt_balance_msg = in_msg_body~load_coins();
    int yt_balance_msg = in_msg_body~load_coins();
    int with_unwrap = in_msg_body~load_uint(1);

    int pt_sum_balance = pt_balance + pt_balance_msg;
    int yt_sum_balance = yt_balance + yt_balance_msg;

    if ((pt_sum_balance > 0) & (yt_sum_balance > 0)) {
        send_finalize_redeem(admin_addr, recipient_addr, msg_value, pt_sum_balance,
yt_sum_balance, query_id, with_unwrap);
        pt_sum_balance = 0;
        yt_sum_balance = 0;
    }

    save_data(admin_addr, recipient_addr, pt_sum_balance, yt_sum_balance);
    return ();
}
```

Figure 3.2: The `register_jetton` operation handler in the `RedeemDeposit` contract
`contracts/YT/redeem/deposit.fc#L83-L99`

The `finalize_redeem` operation handler of the `YTMinter` contract compares the PT and YT amounts transferred by the user, redeems the SY tokens for a minimum of the PT and YT token amounts, and transfers the remaining amount of PT or YT tokens back to the user.

The lack of the token transfer initiator in the `RedeemDeposit` contract's `init_state` allows anyone to send a `register_jetton` message to a user's existing `RedeemDeposit` contract by specifying the user as the `recipient_addr` of the `redeem` message. An attacker can use this to complete a user's redemption by transferring only 1 nano ton of YT or PT token. The user gets back the remaining tokens and needs to transfer them back to the `YTMinter` contract to re-initiate the redemption process. Thus, an attacker can prevent or delay the redemption of the YT and PT tokens to grieve the users by setting up a bot to complete the target user's redemptions.

Exploit Scenario

Alice transfers 1e9 nano ton PT tokens to the `YTMinter` contract with the `redeem` operation and her own address as the recipient in the `forward_payload`. Now, before Alice transfers her YT tokens, Eve transfers 1 nano ton YT token to the `YTMinter` with the `redeem` operation and Alice's address as the recipient in the `forward_payload`. Eve's

`redeem` message sends a `register_jetton` message to Alice's `RedeemDeposit` contract, which redeems only 1 nano ton YT and PT for Alice and returns the remaining 999999999 nano ton YT tokens to Alice.

Recommendations

Short term, include the address of the PT or YT token sender address in the `RedeemDeposit` contract's `init_state` to make it unique for a combination of every `YTMinter`, redeemer, and recipient.

Long term, create a system state specification with the state transition diagrams to document all the valid system states, specifically the intermediate temporary states. Follow the specification to ensure correct access control for all of the state transition functions.

4. Users can lose funds because of incorrect SY token configurations in the YTMinter contract

Severity: **Medium**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-FIVA-4
Target: `contracts/YT/maintance.fc`	

Description

The `YTMinter` contract owner can update the storage values of the `sy_minter_address` and `sy_wallet_address` variables using two different operations. Not keeping these variables in sync can result in users losing funds.

The `YTMinter` contract allows the owner to update the `sy_minter_address` with the `change_sy_minter_address` operation:

```
if (op == op::change_sy_minter_address) {
    check_sender_is_owner(sender_address);
    storage::sy_minter_address = in_msg_body~load_msg_addr();
    save_data();
    throw(successExitCode);
}
```

Figure 4.1: The admin function to update the SY minter configuration
contracts/YT/maintance.fc#L135-L140

The `YTMinter` contract also allows the owner to update the `sy_wallet_address` with the `change_jetton_addresses` operation:

```
if (op == op::change_jetton_addresses) {
    check_sender_is_owner(sender_address);
    cell new_jetton_addresses = in_msg_body~load_ref();
    slice jetton_addresses = new_jetton_addresses.begin_parse();
    if (jetton_addresses.slice_empty?() == 0) {
        storage::sy_wallet_address = jetton_addresses~load_msg_addr();
        storage::pt_minter_address = jetton_addresses~load_msg_addr();
        storage::pt_wallet_address = jetton_addresses~load_msg_addr();
        save_data();
    }
    throw(successExitCode);
}
```

Figure 4.2: The admin function to update Jetton wallet addresses
contracts/YT/maintance.fc#L38-L49

The SYMinter and SYWallet contracts are interconnected contracts, and their stored values in the YTMinter contracts must remain in sync. If the owner updates the sy_minter_address without updating the sy_wallet_address value, users can lose funds from some actions.

Specifically, when a user transfers their underlying asset to the SYMinter contract with the wrap_and_mint_pt_yt operation, then the SY tokens are transferred to the YTMinter contract with the mint_pt_yt operation. If the user warps and mints from a matured YTMinter, and the YTMiner owner updates the st_minter_address in the time between the internal_transfer message reaches the YT minter contract's SYWallet contract and the transfer_notification message reaches the YTMinter contract, then the SY tokens are sent back to the updated sy_minter_address. This will lead to loss of funds because the updated SYMinter contract will not authorize the SYWallet associated with the old SYMinter contract.

Exploit Scenario

The YTMinter owner updates the sy_minter_address but does not update the sy_wallet_address. A user tries to mint a mature YT token and loses her funds.

Recommendations

Short term, update the owner functions to update the sy_minter_address and sy_wallet_address from a single function.

Long term, review all the admin functions to ensure that token configurations remain in sync and other configuration changes do not break the system state.

5. The SYWallet contract is not tested

Severity: Informational	Difficulty: Low
Type: Testing	Finding ID: TOB-FIVA-5
Target: `tests/utils/setup.ts`	

Description

The `SYWallet` contract is not tested because the `syWalletCode` variable is overwritten in the `setup.js`. Only the `EvaaSyWallet` contract is used for both generic and Evaa SY token contract tests.

```
async function compileContracts() {
    underlyingMinterCode = await compile('StakeMinter');
    underlyingWalletCode = await compile('StakeWallet');
    syMinterCode = await compile('SYMinter');
    syWalletCode = await compile('SYWallet');
    ptMinterCode = await compile('PTMinter');
    ptWalletCode = await compile('PTWallet');
    ytMinterCode = await compile('YTMinter');
    ytWalletCode = await compile('YTWallet');
    redeemDepositCode = await compile('RedeemDeposit');
    syWalletCode = await compile('EvaaSYWallet');
    evaaSYMinterCode = await compile('EvaaSYMinter');
    evaaSYWalletCode = await compile('EvaaSYWallet');
    evaaMockCode = await compile('EvaaMock');
}
```

Figure 5.1: The test setup function `tests/utils/setup.ts#L39-L53`

Recommendations

Short term, update the test setup file to use a different variable for the `EvaaSYWallet` contract.

Long term, ensure that the old code coverage is not affected by the new code and new test cases.

6. Lack of a gas check in the wrap operation handler

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-FIVA-6
Target: `contracts/SY/generic/minter.fc`	

Description

The lack of a gas check in the `wrap` operation handler of the `SYMinter` contract allows users to send just enough gas to increase the `total_supply` but not increase the receiver's SY token balance by triggering an out-of-gas error in the `SYWallet` contract. This leads to an imbalance between the `total_supply` and the sum of all user balances.

```
if (fwd_op == op::wrap) {
    ;; check if tokens come from the proper wallet
    throw_unless(error::invalid_underlying, equal_slice_bits(sender_address,
storage::underlying_address));

    raw_reserve(get_storage_gas_units(time_delta::day) * gas_unit_price, 4);

    ;; if we have max_total_supply configured, we should check if we can mint more
tokens
    if ((storage::max_total_supply != 0) & (storage::total_supply + jetton_amount >
storage::max_total_supply)) {
        ;; return jettons back
        transfer_jettons(
            storage::underlying_address,
            from_address,
            from_address,
            jetton_amount,
            0,
            query_id,
            CARRY_REMAINING_BALANCE,
            0,
            null()
        );
        return ();
    }
    mint_tokens(to_address, jetton_amount, query_id, 0, null(), to_address,
storage::jetton_wallet_code, CARRY_REMAINING_BALANCE);
    storage::total_supply += jetton_amount;
    save_data();
    return ();
}
```

Figure 6.1: The `wrap` message handler `contracts/SY/generic/minter.fc#L142-L169`

Recommendations

Short term, add a gas check to the `wrap` operation handler of the `SYMinter` contract to ensure that the `wrap` operation consistently updates the whole system state.

Long term, review the whole codebase to detect lack of gas checks or insufficient gas checks that can lead to an inconsistent system state.

7. Curve stable swap AMM is not usable

Severity: **Informational**	Difficulty: **Low**
Type: Configuration	Finding ID: TOB-FIVA-7
Target: `contracts/AMM/markets/curve_stable/math.fc`	

Description

The `AMM/markets/curve_stable/math.fc` uses `CUBE_STABLE` as `SWAP_TYPE` instead of `CURVE_STABLE_SWAP`. The use of an incorrect swap type makes it unusable.

The FIVA `Pool` contract uses the `SWAP_TYPE` constant to modify the contract storage usage for different types of automated market-making (AMM) mechanisms. The protocol supports the constant product, cube stable swap, and curve stable swap AMMs. Each AMM has its own directory in the `markets` directory with a `math.fc` file in it. These `math.fc` files contain the AMM-specific invariant and swap functions along with the `SWAP_TYPE` constant definition.

The `SWAP_TYPE` value is used in the `load_data` and `store_data` functions of the `storage.fc` file:

```
if (swap_type == CURVE_STABLE_SWAP) {
    storage::amplification_coefficient = ds~load_uint(8);
}
if (swap_type == CUBE_STABLE) {
    storage::sy_pt_price = ds~load_uint(32);
}
```

Figure 7.1: A snippet of the `load_data` function of the `Pool` contract
`contracts/AMM/storage.fc#L83-L88`

However, the `math.fc` file of the curve stable swap has the `CUBE_STABLE` value for the `SWAP_TYPE` constant instead of the `CURVE_STABLE_SWAP`, as shown in the figure below:

```
const SWAP_TYPE = CUBE_STABLE;
```

Figure 7.2: The `SWAP_TYPE` constant definition
`contracts/AMM/markets/curve_stable/math.fc#L6-L6`

The incorrect `SWAP_TYPE` value results in the `storage::amplification_coefficient` value not being assigned, which makes the curve stable swap AMM unusable because of division by zero in all of the functions of the `AMM/markets/curve_stable/math.fc` file.

The curve stable swap AMM is deprecated; therefore, the severity of this issue has been downgraded to informational.

Exploit Scenario

The FIVA administrator deploys a new curve stable pool by updating the `swaps.fc` to include the `curve_stable/math.fc` and using the curve stable pool configuration. However, the `SWAP_TYPE` constant remains incorrect in the `curve_stable/math.fc`, which makes the newly deployed pool unusable.

Recommendations

Short term, update the `AMM/markets/curve_stable/math.fc` file to assign the `CURVE_STABLE_SWAP` value to the `SWAP_TYPE` constant.

Long term, improve the test suite to test all code paths and all contracts.

8. An incorrect balance check for the PT-to-SY swap can lead to a loss of funds

Severity: **High**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-FIVA-8
Target: `contracts/AMM/markets/swaps.fc`	

Description

The AMM contract fails to account for locked SY tokens when validating PT-to-SY swaps, which can lead to users losing their PT tokens without receiving any SY tokens in return.

In the FIVA protocol, users can swap between different token types (SY, PT, and YT) through the AMM pool. During certain operations, some tokens may be temporarily locked in the pool. The protocol tracks these locked tokens using the `storage::sy_locked` and `storage::pt_locked` state variables.

When a user attempts to swap PT for SY tokens, the `handle_swap_pt_for_sy` function should verify that the pool has enough available (non-locked) SY tokens to fulfill the swap. However, unlike other swap functions in the codebase, this function checks against only the total SY supply without accounting for locked tokens:

```
if ((sy_out < max(min_sy_out, MIN_SWAP_AMOUNT)) | (sy_out >
(storage::total_supply_sy - MINIMUM_LIQUIDITY))) {
```

Figure 8.1: The balance check in the PT to SY swaps `contracts/AMM/markets/swaps.fc#75`

In contrast, other swap functions correctly account for locked tokens. For example, the `handle_swap_sy_for_yt` function includes `storage::sy_locked` in its validation:

```
(sy_out > (storage::total_supply_sy - storage::sy_locked - MINIMUM_LIQUIDITY))
```

Figure 8.2: The balance check in the SY to YT swaps `contracts/AMM/markets/swaps.fc#124`

This inconsistency creates a problematic scenario when locked SY tokens are present in the pool. The swap validation may incorrectly pass because it checks against only the total SY supply, not the available supply. When the transaction proceeds to actually transfer SY tokens, it will fail due to an insufficient available balance. This would result in the user permanently losing their PT tokens without receiving any SY tokens in return, while leaving the pool in an inconsistent state where its accounting and actual token balances are out of sync, potentially affecting future operations.

Exploit Scenario

A user attempts to swap 200 PT tokens for SY in a pool where a significant amount of SY tokens are locked. The pool has a total of 1,000 SY tokens, but 900 of these are locked, leaving only 100 SY tokens actually available. The validation check passes incorrectly because it only compares against the total supply of SY tokens (1,000), not the available supply (100). When the transaction proceeds and the pool's state is updated, the user's 200 PT tokens are added to the pool, but the transfer of 200 SY tokens fails due to insufficient available balance. As a result, the user loses their PT tokens without receiving any SY tokens, and the pool's accounting becomes inconsistent with its actual token balances.

Recommendations

Short term, modify the validation check in `handle_swap_pt_for_sy` to account for locked SY tokens.

Long term, expand the test suite to specifically verify swap behavior when tokens are locked in the pool.

9. An attacker can grieve users by completing their liquidity provision operation

Severity: **High**	Difficulty: **Medium**
Type: Undefined Behavior	Finding ID: TOB-FIVA-9
Target: `contracts/AMM/markets/pool.fc`	

Description

An attacker can complete a user's pool liquidity-providing action by transferring an insignificant amount to the pool with the `add_liquidity` message, mentioning the same `receiver` as the user. This causes the user to receive significantly fewer LP tokens than expected.

The FIVA protocol allows users to provide liquidity to the SY/PT pool. The liquidity provision process requires users to transfer both SY and PT tokens to the `Pool` contract with a `forward_payload`, including an `add_liquidity` operation. On the first token transfer, a `Deposit` contract is deployed to store the `Pool` address, recipient address, and the amount of SY or PT transferred by the user. On the second token transfer, the `Deposit` contract checks if both SY and PT are deposited and a slippage parameter is provided, then sends a `provide_lp` message to the `Pool` contract to mint LP tokens to the recipient.

```
 if (op == op::add_liquidity) {
    sy_balance += in_msg_body~load_coins();
    pt_balance += in_msg_body~load_coins();
    int min_lp_out = in_msg_body~load_coins();
    int with_unwrap = in_msg_body~load_uint(1);

    ;; TODO: consider to use min balance limit (i.e. 1000 instead of 0)
    if ((min_lp_out > 0) & (sy_balance > 0) & (pt_balance > 0)) {
        send_provide_lp(owner_addr, recipient_addr, sy_balance, pt_balance,
min_lp_out, query_id, with_unwrap);
        sy_balance = 0;
        pt_balance = 0;
    }

    save_data(owner_addr, recipient_addr, sy_balance, pt_balance);
    return ();
}
```

Figure 9.1: The `add_liquidity` operation handler in the `Deposit` contract `contracts/AMM/lp_deposit/deposit.fc#77-92`

However, an attacker can exploit this process by making the second transfer before the legitimate user, sending a negligible amount of tokens while specifying the same receiver as the user and an arbitrary value for `min_lp_out`. Because of the same pool and `receiver` combination, the attacker's `add_liquidity` message is forwarded to the user's `Deposit` contract, which then sends a `provide_lp` operation message to the pool. Since the amounts of PT and SY are imbalanced, the user will only receive the LP tokens minted by the minimum of the PT and SY tokens while losing all the tokens deposited in the first token transfer. The slippage protection is bypassed because the `attacker` controls the `min_lp_out` parameter.

Exploit Scenario

An attacker exploits a vulnerability in the two-phase deposit process by targeting users who have deposited 10,000 SY tokens into the `Pool` contract but have not yet deposited their PT tokens. After identifying a victim's `Deposit` contract, the attacker sends a negligible amount of PT tokens, just enough to mint 1 nanoton LP token, to the pool while setting the victim as the recipient and specifying an extremely low `min_lp_out` value of 1, to bypass the slippage protection.

When the `Deposit` contract processes the `add_liquidity` operation, it executes the `send_provide_lp` function using the attacker's manipulated parameters. The victim receives 1 nanoton LP token while losing their entire 10,000 SY token deposit.

Recommendations

Short term, include the address of the SY or PT token sender address in the `Deposit` contract's `init_state` to make it unique for a combination of every `Pool`, recipient, and token sender.

Long term, create a system state specification with the state transition diagrams to document all the valid system states, specifically the intermediate temporary states. Follow the specification to ensure correct access control for all of the state transition functions.

10. An integer overflow in the cube stable market invariant calculation can make the AMM unusable for swaps

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-FIVA-10
Target: `contracts/AMM/markets/cube_stable/math.fc`	

Description

The cube stable market AMM's invariant calculation can overflow when performing cube operations on large token supplies, exceeding TON's integer limits. The overflow will result in users losing their funds in an attempt to swap any tokens from the pool.

This market AMM uses an invariant function that performs cube operations on token supplies, creating a significant risk of integer overflow. The `invariant` function is shown in figure 10.1:

```
(int) invariant() inline {
    int res = math::cube(storage::total_supply_sy * storage::sy_pt_price / DIVIDER)
* storage::total_supply_pt +
            (storage::total_supply_sy * storage::sy_pt_price / DIVIDER) *
math::cube(storage::total_supply_pt);
    return res;
}
```

Figure 10.1: Invariant calculation for cube stable market (contracts/AMM/markets/cube_stable/math.fc#11–15)

This formula involves raising token supplies to the power of 3 and multiplying large numbers together, which can easily exceed the 256-bit integer capacity used in TON. This overflow will result in transaction failures.

In TON, integers have a maximum value of $2^{256 - 1}$. If both SY and PT tokens have similar supplies and assuming `sy_pt_price` ≈ `DIVIDER`, the invariant formula can be simplified to the following:

```
token_amount^3 * token_amount + token_amount * token_amount^3
2 * token_amount^4 < 2^256
token_amount < 10^19
```

Figure 10.2: Max token supply calculation

For tokens with nine decimals, `real_world_amount < 10^10`, meaning if 10 billion SY and PT tokens are provided to the pool, an overflow can occur on the invariant calculation. Since the SY and PT tokens will always have nine decimals, the impact is limited to the case where the supply reaches 10 billion tokens. Users can withdraw their liquidity to bring the token supply below the 10 billion limit to make the pool swaps usable again.

Exploit Scenario

Eve, an attacker, adds a calculated amount of liquidity that pushes the pool's total supply close to 10 billion. When Alice, a victim, attempts to swap PT to SY tokens, she sends her PT tokens to the pool. The invariant calculation in the pool contract overflows and traps Alice's PT tokens in the pool contract. Alice has already sent PT tokens but has not received SY tokens; she has lost her PT tokens.

Recommendations

Short term, document this limitation and deploy a system to monitor pool reserves reaching the 10^{10} limit. Such a system will help prevent fund loss to users.

Long term, conduct a thorough analysis of all mathematical overflows possible in the codebase. Based on the analysis results, implement overflow protection for identified critical paths, implement transaction revert handling, and update documentation with any additional discovered limitations.

11. The YTMinter contract's get_claimable_interest function deducts the protocol fee twice

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-FIVA-11
Target: `contracts/YT/getters.fc`	

Description

The `get_claimable_interest` function of the `YTMinter` contract returns the wrong interest amount by deducting the protocol fee twice, resulting in users expecting less than the actual accrued interest.

Users use the `get_claimable_interest` function of the `YTMinter` contract to estimate the accrued interest on their YT tokens. They can use this estimated amount to transfer the earned tokens to another contract. Users call the `get_claimable_interest` function with their YT token balance, `last_collected_index`, and `acquired_amount` stored in their `YTWallet` contract. The `get_claimable_interest` function executes the `get_acquired_interest` function to compute the newly accrued interest after the last claim action by the user and adds it to the `acquired_amount` to compute the total interest:

```
(int, int) get_claimable_interest(int yt_amount, int last_collected_index, int
acquired_amount) method_id {
    load_data();
    int acquired_interest_sy = get_acquired_interest(yt_amount,
last_collected_index);

    int interest = acquired_interest_sy + acquired_amount;
    (int protocol_fee, int interest_wo_fee) = calc_protocol_fee(storage::fee,
DIVIDER, interest);
    return (interest_wo_fee, protocol_fee);
}
```

Figure 11.1: The `get_claimable_interest` function of the `YTMinter` contract
`contracts/YT/getters.fc#L111-L118`

The `YTWallet` contract sends a `request_acquired_update` message to the `YTMinter` contract to get and store the interest acquired by the user at the time of the YT token transfer. The `YTMinter` contract computes the newly acquired interest and deducts the protocol fee before sending the interest amount to the `YTWallet` contract to store, as shown in figure 11.2:

```
int acquired_interest_sy = get_acquired_interest(user_yt_balance,
last_collected_interest_index);

if(acquired_interest_sy > 0){
    ;; debt to the user in a form of interest
    acquired_interest_sy = take_protocol_fee(acquired_interest_sy);
}

raw_reserve(fee::min_tons_for_storage_min, 4);
;; message to the wallet to update index
update_wallet_index_and_acquired(
    storage::index,
    acquired_interest_sy,
    sender_address,
    0,
    CARRY_REMAINING_BALANCE
);
```

Figure 11.2: A snippet of the `request_acquired_update` operation handler
`contracts/YT/minter.fc#L90-L105`

However, the `get_claimable_interest` function deducts the protocol fee from the total interest amount, which includes the `acquired_interest` stored in the `YTWallet` contract after deducting the protocol fee. Thus, the `get_claimable_interest` function deducts the protocol fee twice on the user's stored `acquired_interest` and returns an incorrect interest amount.

Recommendations

Short term, correct the `get_claimable_interest` function to deduct the protocol fee only from the `acquired_interest_sy` value instead of the total interest amount.

Long term, expand the test suite by adding test cases for the getter functions to check that they return correct values.

12. Incorrect forward value when minting PT in function mint_py_jettons

Severity: **Low**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-FIVA-12
Target: `contracts/YT/mint.fc`	

Description

The `mint_py_jettons` function of the `YTMinter` contract assigns an incorrect forward value when minting PT jettons, resulting in excessive gas consumption.

The `mint_py_jettons` function handles the minting process for both PT and YT jettons. As part of its logic, it calculates the forward value that will be included in the transfer notification message after minting PT and YT tokens by calling the `fee::mint_pt` and `fee::mint_yt` functions, respectively.

```
;; mint PT and YT jettons
(int ) mint_py_jettons(int jetton_amount, slice pt_recipient_address, slice
yt_recipient_address, int query_id, int fwd_fee, slice fwd_cs) impure {
    (int pt_msg_value, int pt_mint_value, int pt_fwd_value) = fee::mint_pt(fwd_fee);
    (_, int yt_fwd_value) = fee::mint_yt(fwd_fee);

    cell pt_fwd_payload = fwd_cs.slice_empty?() ? null() : fwd_cs~load_maybe_ref();
    cell yt_fwd_payload = fwd_cs.slice_empty?() ? null() : fwd_cs~load_maybe_ref();
    pt_fwd_value = cell_null?(pt_fwd_payload) ? 0 : yt_fwd_value;
    yt_fwd_value = cell_null?(yt_fwd_payload) ? 0 : yt_fwd_value;

    int amount_to_mint = jetton_amount * storage::index / index_precision;
    mint_pt(pt_recipient_address, amount_to_mint, query_id, pt_msg_value,
pt_mint_value, pt_fwd_value, pt_fwd_payload, PAY_FEES_SEPARATELY);
    mint_yt(yt_recipient_address, amount_to_mint, query_id, 0, yt_fwd_value,
yt_fwd_payload, CARRY_REMAINING_BALANCE);
    return amount_to_mint;
}
```

Figure 12.1: The `mint_py_jettons` function in `contracts/YT/mint.fc`

Later in the function, a check determines whether the forward payload is empty. If so, the forward value is overridden to 0; otherwise, it should remain unchanged. However, the value of `pt_fwd_value` is mistakenly set to `yt_fwd_value` instead of keeping its original value. Since `yt_fwd_value` is larger than `pt_fwd_value`, the PT transfer notification ends up using a higher forward value than necessary, leading to wasted gas.

Recommendations

Short term, correct the assignment of `pt_fwd_value` to ensure that it retains its intended value and is not overwritten by `yt_fwd_value`.

Long term, develop test cases specifically aimed at validating gas efficiency and correctness in token minting workflows across all paths.

13. Race condition in YT swap and index update can lead to loss of funds

Severity: **High**	Difficulty: **High**
Type: Timing	Finding ID: TOB-FIVA-13
Target: `contracts/AMM/markets/swaps.fc`	

Description

A race condition between the YT token swap messages and the `Pool` and `YTMinter` contract index update messages can lead to loss of funds.

The FIVA `Pool` contract allows users to swap SY to YT tokens. The `Pool` contract does not hold YT tokens as liquidity; therefore, it mints new PT/YT tokens for swapping SY to YT. When a user transfers SY tokens with the `swap_sy_for_yt` message, then the `Pool` contract computes the amount of SY tokens needed to mint new PT/YT tokens:

```
;; sy_out is equal to amount of YT which will be sent to the user
(int sy_out, int protocol_fee) = swap_exact_sy_for_yt(jetton_amount, inv_before);
int yt_out = sy_out * storage::index / DIVIDER;

;; reserve amount of tons we had before message processing to prevent pool draining
raw_reserve(fee::min_tons_for_pool_storage, 4);

if (
    (yt_out < max(min_yt_out, MIN_SWAP_AMOUNT)) |
    (sy_out > (storage::total_supply_sy - storage::sy_locked - MINIMUM_LIQUIDITY))
) {
    send_sy_back(from_address, recipient_addr, jetton_amount, query_id,
with_unwrap);
} else {
    storage::total_supply_sy -= (sy_out + protocol_fee);
    storage::total_supply_sy += jetton_amount;
    storage::collected_sy_protocol_fee += protocol_fee;
    ;; Pool will receive PT amount equal to YT_out in some time from YT minter
    storage::total_supply_pt += yt_out;
    storage::pt_locked += yt_out;
    ;; If swap decreases pool invariant, don't commit it and return jettons back
    if inv_before > invariant() {
        send_sy_back(from_address, recipient_addr, jetton_amount, query_id,
with_unwrap);
    } else {
        int fwd_amount = fee::mint_pt_yt(fee::avg_fwd_fee);
        mint_py(sy_out, my_address(), recipient_addr, 0, fwd_amount, query_id,
CARRY_REMAINING_BALANCE);
        save_data(SWAP_TYPE);
```

```
    }
}
```

Figure 13.1: A snippet of the swap_sy_for_pt message handler contracts/AMM/markets/swaps.fc#L115-L142

The Pool contract then computes the yt_out amount by dividing the computed sy_out amount by the storage::index value stored in the Pool contract. This yt_out is the expected amount of the YT and PT tokens that will be minted from the YTMinter contract by transferring out the sy_out amount of SY tokens. The Pool contract adds yt_out to the storage::total_supply_pt and storage::pt_locked values. The Pool contract transfers the sy_out amount of SY tokens to the YTMinter contract with the mint_pt_yt message in the forward_payload.

The YTMinter contract receives the SY tokens and computes the amount of PT/YT tokens to mint by multiplying the storage::index values stored in the YTMinter contract with the amount of SY tokens transferred. The YTMinter contract mints new YT tokens for the user and new PT tokens for the Pool contract. The YTMinter contract sends the mint_notification message with the PT tokens to the Pool contract to allow the Pool contract to update the storage::pt_locked value.

```
storage::pt_locked -= min(jetton_amount, storage::pt_locked);
```

Figure 13.2: contracts/AMM/markets/swaps.fc#L313-L313

However, if the storage::index values stored in the Pool contract and the YTMinter contract are not the same, then the yt_out computed by the Pool contract will differ from the amount of PT tokens minted by the YTMinter contract. If the index is higher in the YTMinter than the Pool contract, then a higher-than-expected amount of PT tokens will be minted to the Pool contract, which will remain stuck in the Pool contract because they are not added to the storage::total_supply_pt value. A race condition between the swap message and the index update message to the YTMinter and Pool contract can lead to the storage::index value difference and consequently a loss of funds.

This issue also affects the YT to SY token swaps, with a difference in the expected amount of redeemed SY tokens and the redeemed amount of the SY tokens.

Exploit Scenario

The FIVA protocol administrator wants to update the index value stored in YTMinter and Pool contract from 1.1 to 1.2. The administrator sends the update_minter_index message to the YTMinter contract and then sends the update_pool_index message to the Pool contract. The YTMinter contract index is updated, but before the update_pool_index messages reaches the Pool contract, it receives a swap_sy_for_yt message to swap 1,000 SY tokens to YT tokens. The Pool contract expects the YTMinter contract to mint 1,100 PT tokens, but it mints 1,200 PT tokens. The Pool contract adds only

1,100 PT tokens to the pool liquidity, and the remaining 100 PT tokens are stuck in the `Pool` contract.

Recommendations

Short term, send the `storage::index` value from the `Pool` contract to the `YTMinter` contract and from the `YTMinter` contract to the `Pool` contract. Additionally, update the pool reserves, considering changes in the `storage::index` value for the YT swap operations.

Long term, analyze the codebase to document the assumptions that a value will remain in sync between multiple contracts, validate these syncing assumptions, and consider out-of-sync values when implementing features and test cases.

14. Lack of validation checks in admin action handlers	
Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-FIVA-14
Target: `contracts/*.fc`	

Description

All smart contracts implement functions to update their configuration parameters and storage values. These functions lack data validation checks, allowing privileged actors to manipulate them to steal funds from protocol users.

The following action handler functions lack validation checks:

- The `update_protocol_fee` action handler of the `YTMinter` contract allows the owner or maintainer to update the protocol fee to up to 4294%.
- The `withdraw_jettons` action handler of the `YTMinter` contract allows the owner to withdraw any amount of any jetton from the contract.
- The `mint_pt` action handler of the `YTMinter` contract allows the owner to mint any amount of PT tokens.
- The `update_lp_fee` action handler of the `Pool` contract lacks an upper bound check on the provided lp fee.
- The `update_ref_fee` action handler of the `Pool` contract lacks an upper bound check.

Recommendations

Short term, add the validation checks described above to ensure that all configuration values and contract parameters have upper and lower bounds and validation checks.

Long term, expand the test suite by adding cases to detect incorrect configurations.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage
Transaction Ordering	The system's resistance to transaction-ordering attacks

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category does not apply to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Code Quality Recommendations

The following recommendations are not associated with any specific vulnerabilities. However, they will enhance code readability and may prevent the introduction of vulnerabilities in the future.

- The `math::sqrt` function in the math library performs 15 iterations of the Newton-Raphson method, though mathematical analysis shows that only eight iterations are sufficient for full precision with 256-bit numbers. This results in unnecessary computation without accuracy gains. To improve efficiency, the iteration count should be reduced, and the initial estimate of `r` should be refined before iteration to enhance precision.
- The EVAA and YT token storage declarations contain storage named “maturiry” (with a typo), but the comment and the variable purpose refer to “maturity.” This name should be corrected to improve code readability and prevent confusion.

D. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

From May 26 to May 28, 2025, Trail of Bits reviewed the fixes and mitigations implemented by the FIVA team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the 14 issues described in this report, FIVA has resolved 12 issues and has not resolved the remaining two issues. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Severity	Status
1	Lack of a two-step process for critical operations	Low	Unresolved
2	Lack of validation checks in the upgrade_storage operation handler	High	Resolved
3	An attacker can prevent the redemption of YT and PT tokens from the YTMinter contract	High	Resolved
4	Users can lose funds because of incorrect SY token configurations in the YTMinter contract	Medium	Resolved
5	The SYWallet contract is not tested	Informational	Resolved
6	Lack of a gas check in the wrap operation handler	Informational	Resolved
7	Curve stable swap AMM is not usable	Informational	Unresolved
8	An incorrect balance check for the PT-to-SY swap can lead to a loss of funds	High	Resolved
9	An attacker can grieve users by completing their liquidity provision operation	High	Resolved

10	An integer overflow in the cube stable market invariant calculation can make the AMM unusable for swaps	Informational	Resolved
11	The YTMinter contract's get_claimable_interest function deducts the protocol fee twice	Informational	Resolved
12	Incorrect forward value when minting PT in function mint_py_jettons	Low	Resolved
13	Race condition in YT swap and index update can lead to loss of funds	High	Resolved
14	Lack of validation checks in admin action handlers	Informational	Resolved

Detailed Fix Review Results

TOB-FIVA-1: Lack of a two-step process for critical operations
Unresolved.

The client provided the following context for this finding's fix status:

> *We are currently using multisig wallet with 2 out of 3 configuration. 2 step process will be implemented later in the scope of improved decentalization.*

TOB-FIVA-2: Lack of validation checks in the upgrade_storage operation handler
Resolved in PR #124 and PR #168. The `maintance.fc` files imported by the `Pool`, `SYMinter`, `EvaaSYMinter`, `PTMinter`, and `YTMinter` contracts now implement the `validate_storage_cell` function to validate the new storage cell provided by the administrator. The `UnlimitedMinter`, `UnlimitedMinterV2`, and `Collection` contracts do not need to implement the new storage cell validation check because they are not part of the production deployment.

TOB-FIVA-3: An attacker can prevent the redemption of YT and PT tokens from the YTMinter contract
Resolved in PR #126. The `RedeemDeposit` contract now includes the redemption initiator user's address in the `init_state` and stores it as the `requester_addr` variable. Additionally, the `RedeemDeposit` contract validates the token sender to be the same as the stored `requester_addr` in the `register_jetton` action handler.

TOB-FIVA-4: Users can lose funds because of incorrect SY token configurations in the YTMinter contract
Resolved in PR #153. The `YTMinter` contract now implements an action to update the SY minter and SY wallet address together, and another action to update the PT minter and PT wallet address together.

TOB-FIVA-5: The SYWallet contract is not tested
Resolved in commit 9744488. The `syWalletCode` variable overwriting statement has been removed from the test setup function.

TOB-FIVA-6: Lack of a gas check in the wrap operation handler
Resolved in PR #152. The `wrap` action handler of the `SYMinter` contract now checks that the gas is enough to complete the SY token transfer to the user's wallet.

TOB-FIVA-7: Curve stable swap AMM is not usable
Unresolved.

The client provided the following context for this finding's fix status:

> *We are not going to use Curve stable swap AMM in the near future*

TOB-FIVA-8: An incorrect balance check for the PT-to-SY swap can lead to a loss of funds
Resolved in PR #136. The `sy_out` validation check in the `handle_swap_pt_for_sy` function now deducts the locked SY amount from the total SY supply to compute the available SY token balance.

TOB-FIVA-9: An attacker can grieve users by completing their liquidity provision operation
Resolved in PR #127. The `Deposit` contract now includes the liquidity depositor user's address in the `init_state` and stores it as the `requester_addr` variable.

TOB-FIVA-10: An integer overflow in the cube stable market invariant calculation can make the AMM unusable for swaps
Resolved in PR #156 and PR #169. The `Pool` contract now limits the maximum amount of SY and PT tokens to be less than 10^{19} nanotons in all of the YT swap functions and the liquidity addition function.

TOB-FIVA-11: The YTMinter contract's get_claimable_interest function deducts the protocol fee twice
Resolved in PR #134. The `get_claimable_interest` function now correctly deducts the protocol fee from the newly acquired interest only.

TOB-FIVA-12: Incorrect forward value when minting PT in function mint_py_jettons
Resolved in PR #154. The `mint_py_jettons` function now correctly assigns the value of the `pt_fwd_value` variable.

TOB-FIVA-13: Race condition in YT swap and index update can lead to loss of funds
Resolved in PR #155 and PR #170. The `Pool` contract now sends its `storage::index` value with the messages to the `YTMinter` contract. The `YTMinter` contract uses the `index` value sent by the `Pool` contract instead of the value stored in its contract storage to mint to redeem the same amount of tokens as expected by the `Pool` contract.

An attacker can exploit the use of the `Pool` contract `index` value in the `YTMinter` contract to get a higher amount from swaps when the `index` value in both contracts is different. However, such an exploit is extremely difficult because it can be done only in a narrow time window when the `index` value is different in both contracts and requires timing a series of transactions in the TON blockchain. Additionally, this issue has a limited impact because the `index` value changes by an insignificant amount with every update.

TOB-FIVA-14: Lack of validation checks in admin action handlers
Resolved in PR #157. The `YTMinter` and `Pool` contracts now implement validation checks to ensure that the configurable values have an upper and lower bound.

E. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries and government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on X and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

Trail of Bits performed all activities associated with this project in accordance with a statement of work and an agreed-upon project plan.

Security assessment projects are time-boxed and often rely on information provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test software controls and security properties. These techniques augment our manual security review work, but each has its limitations. For example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. A project's time and resource constraints also limit their use.