FIVA Evaa Integration

Security Assessment (Summary Report)

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

Project Targets

The engagement involved reviewing and testing the following target.

contracts_v2

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

Executive Summary

Engagement Overview

FIVA engaged Trail of Bits to review the security of the Evaa protocol integration with their yield tokenization protocol. The integration implements a custom `EvaaSYMinter` contract to tokenize the Evaa deposit position.

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, and on identifying race conditions among user actions, corruption of the 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 well structured 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 identified two high-severity issues and one informational issue arising from insufficient access control checks (TOB-FIVAEVAA-1, TOB-FIVAEVAA-2, TOB-FIVAEVAA-3). We also found a timing vulnerability (TOB-FIVAEVAA-4) that allows attackers to exploit the interest rate update lag in the `EvaaSYMinter` contract.

Recommendations

Based on the findings identified during the security review, Trail of Bits recommends that Ritual take the following steps to secure the system:

- **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.

Summary of Findings

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

ID	Title	Type	Severity
1	Insufficient access control checks in the supply_excess message handler of the EvaaSYMinter contract	Access Controls	**High**
2	Insufficient access control checks in the supply_fail_excess message handler of the EvaaSYMinter contract	Access Controls	**High**
3	Insufficient access control checks in the withdraw_success message handler of the EvaaSYMinter contract	Access Controls	**Informational**
4	Users can benefit by sandwiching the index update message to the EvaaSYMinter contract	Timing	**Low**

Detailed Findings

1. Insufficient access control checks in the supply_excess message handler of the EvaaSYMinter contract	
Severity: **High**	Difficulty: **Low**
Type: Access Controls	Finding ID: TOB-FIVAEVAA-1
Target: `contracts/SY/evaa/minter.fc`	

Description

Insufficient access control checks in the `supply_excess` message handler of the `EvaaSYMinter` contract allow anyone to mint an infinite number of Evaa SY tokens.

The FIVA protocol uses the `EvaaSYMinter` contract to allow users to mint SY tokens for their Evaa deposits. Users transfer their base token to the `EvaaSYMinter` contract with the `wrap` message as the `forward_payload` of the Jetton `transfer` message. The `EvaaSYMinter` contract validates the `transfer_notification` message and transfers the base token to the Evaa master contract with the `supply_master` message as the `forward_payload`. The Evaa master contract updates the user deposit amount and sends a `supply_excess` message to the `EvaaSYMinter` contract.

The `EvaaSYMinter` parses the custom payload included in the `supply_excess` message and validates the message based on values included in the message. After validating the message, the `EvaaSYMinter` contract mints the SY tokens to the `recipient` address mentioned in the `supply_excess` message:

```
if (op == op::supply_excess) {
    slice custom_payload = in_msg_body~load_ref().begin_parse();
    if (custom_payload.slice_empty?()) {
        return ();
    }

    int custom_op = custom_payload~load_op();
    throw_unless(error::invalid_custom_payload, custom_op ==
op::evaa::supply_success);

    int query_id = custom_payload~load_query_id();
    int jetton_amount = custom_payload~load_uint(64);
    slice sender = custom_payload~load_msg_addr();
    slice recipient = custom_payload~load_msg_addr();
    throw_unless(error::invalid_custom_payload, equal_slice_bits(sender,
```

```
my_address()));

    raw_reserve(0, 4);

    mint_tokens(recipient, jetton_amount, query_id, fwd_fee, null(), recipient,
storage::jetton_wallet_code, CARRY_REMAINING_BALANCE);
    storage::total_supply += jetton_amount;
    save_data();
    return ();
}
```

Figure 1.1: The `supply_excess` message handler
`contracts/SY/evaa/minter.fc#L253-L274`

However, the `supply_excess` message handler does not check that the sender is the Evaa master contract. The `supply_excess` message handler only validates the `custom_op` and `sender` values included in the message to be the same as the values set by the `EvaaSYMinter` contract while sending the `supply_master` message to the Evaa master contract. This allows anyone to send a `supply_excess` message, including the `custom_op` as `op::evaa::supply_success` and `sender` as the address of the `EvaaSYMinter` contract, to mint an infinite number of Evaa SY tokens.

Exploit Scenario

Eve sends a `supply_excess` message, including the `jetton_amount` value of 1,000 nano tons, `custom_op` as `op::evaa::supply_success`, and `sender` as the address of the `EvaaSYMinter` contract, to the `EvaaEYMinter` contract to mint 1,000 nano tons of Evaa SY tokens. Eve repeats this process to mint an infinite number of Evaa SY tokens.

Recommendations

Short term, add access control checks to ensure that the Evaa master contract has sent the `supply_excess` message.

Long term, implement access control checks on the message sender address or the Jetton sender address instead of the sender value included in the message. Consider using protocol-controlled values instead of user-controlled values while implementing access control checks.

2. Insufficient access control checks in the supply_fail_excess message handler of the EvaaSYMinter contract	
Severity: **High**	Difficulty: **High**
Type: Access Controls	Finding ID: TOB-FIVAEVAA-2
Target: `contracts/SY/evaa/minter.fc`	

Description

Insufficient access control checks in the `supply_fail_excess` message handler of the `EvaaSYMinter` contract allow anyone to send a `transfer_notification` message to steal underlying assets from the `EvaaSYMinter` contract.

The FIVA protocol uses the `EvaaSYMinter` contract to allow users to mint SY tokens for their Evaa deposits. Users transfer their base token to the `EvaaSYMinter` contract with the `wrap` message as the `forward_payload` of the Jetton `transfer` message. The `EvaaSYMinter` contract validates the `transfer_notification` message and transfers the base token to the Evaa master contract with the `supply_master` message as the `forward_payload`. The Evaa master contract updates the user deposit amount and sends a `supply_excess` message to the `EvaaSYMinter` contract. In case the Evaa base token supply fails, the Evaa master contract transfers the base token back to the `EvaaSYMinter` contract with the `supply_fail_excess` message in the `forward_payload`.

The `EvaaSYMinter` parses the custom payload included in the `supply_fail_excess` message and validates the message based on values included in the message. After validating the message, the `EvaaSYMinter` contract returns the base tokens to the `recipient` address mentioned in the `supply_fail_excess` message:

```
if (fwd_op == op::supply_fail_excess) {
    int fwd_query_id = fwd_cs~load_uint(64);
    slice custom_payload = fwd_cs~load_maybe_ref().begin_parse();
    if (custom_payload.slice_empty?()) {
        return ();
    }

    int custom_op = custom_payload~load_op();
    throw_unless(error::invalid_custom_payload, custom_op ==
op::evaa::supply_success);

    int query_id = custom_payload~load_query_id();
    int jetton_amount = custom_payload~load_uint(64);
    slice sender = custom_payload~load_msg_addr();
```

```
    slice recipient = custom_payload~load_msg_addr();
    throw_unless(error::invalid_custom_payload, equal_slice_bits(sender,
my_address()));

    transfer_jettons(storage::underlying_address, recipient, recipient,
to_6_decimal(jetton_amount), 0, query_id, CARRY_REMAINING_GAS, fwd_fee, null());
    return ();
}
```

Figure 2.1: The supply_fail_excess message handler
contracts/SY/evaa/minter.fc#L207-L225

However, the `supply_fail_excess` message handler does not check that the sender is the base token wallet of the `EvaaSYMinter` contract, and that the base token sender is the Evaa master contract. The `supply_fail_excess` message handler only validates the `custom_op` and `sender` values included in the message to be the same as the values set by the `EvaaSYMinter` contract while sending the `supply_master` message to the Evaa master contract. This allows anyone to send a `transfer_notification` message, including the `custom_op` as `op::evaa::supply_success` and `sender` as the address of the `EvaaSYMinter` contract, to steal the base tokens from the `EvaaSYMinter` contract.

The `EvaaSYMinter` contract does not keep the base tokens in its wallet; it immediately transfers the user deposits to the Evaa master contract while processing the `wrap` message. Similarly, the base tokens transferred by the Evaa master contract as part of the `unwrap` process are immediately transferred to the user by the `EvaaSYMinter` contract. However, a race condition can lead to a `supply_fail_excess` message being executed after the `internal_transfer` message is executed by the base token wallet of the `EvaaSYMinter` contract and before the `transfer_notification` message reaches the `EvaaSYMinter` contract.

Exploit Scenario

Eve sets up a bot to detect base token transfers to the `EvaaSYMinter` contract. The bot detects that Alice initiated a base token transfer and sends the `supply_fail_excess` message to the `EvaaSYMinter`, which is executed before the `transfer_notification` message of Alice's base token transfer. Eve steals Alice's base tokens, and Alice does not get any Evaa SY tokens minted to her.

Recommendations

Short term, add access control checks to ensure that the `supply_fail_excess` message is sent by the base token wallet of the `EvaaSYMinter` contract and that the Evaa master contract has sent the base tokens.

Long term, review all the transfer notification message handlers to ensure that they include correct access control checks. Consider malicious and fake Jetton transfer messages while implementing access control checks for the transfer notification messages.

3. Insufficient access control checks in the withdraw_success message handler of the EvaaSYMinter contract

Severity: **Informational**	Difficulty: **Medium**
Type: Access Controls	Finding ID: TOB-FIVAEVAA-3
Target: `contracts/SY/evaa/minter.fc`	

Description

An attacker can send the `withdraw_success` message to the `EvaaSYMinter` contract to avoid burning the full amount of the Evaa SY tokens, locking them in the contract.

The `EvaaSYMinter` contract allows users to withdraw their base tokens by sending the `unwrap` message with the Evaa SY token transfer as the `forward_payload`. The `EvaaSYMinter` contract sends the `withdraw_master` message to the Evaa master contract. The Evaa master contract updates the user deposit amount and transfers the base tokens with the `withdraw_success` message in the `forward_payload`.

The `withdraw_success` message handler of the `EvaaSYMinter` contract validates the message, burns the Evaa SY tokens transferred by the user, and sends the base tokens to the recipient address specified by the user:

```
if (fwd_op == op::withdraw_success) {
    throw_unless(error::invalid_underlying,
equal_slice_bits(storage::underlying_address, sender_address));
    slice custom_payload = fwd_cs~load_ref().begin_parse();
    if (custom_payload.slice_empty?()) {
        return ();
    }

    int custom_op = custom_payload~load_op();
    throw_unless(error::invalid_custom_payload, custom_op ==
op::evaa::withdraw_success);

    int query_id = custom_payload~load_query_id();
    int jetton_amount = custom_payload~load_uint(64);
    slice sender = custom_payload~load_msg_addr();
    slice recipient = custom_payload~load_msg_addr();
    throw_unless(error::invalid_custom_payload, equal_slice_bits(sender,
my_address()));

    cell state_init = calculate_jetton_wallet_state_init(my_address(), my_address(),
storage::jetton_wallet_code);
    slice my_sy_wallet_address = calc_address(state_init);
```

```
    burn_jetton(my_sy_wallet_address, recipient, to_9_decimal(jetton_amount),
query_id, fee::jetton_burn(fwd_fee));
    transfer_jettons(storage::underlying_address, recipient, recipient,
jetton_amount, 0, query_id, CARRY_REMAINING_GAS, fwd_fee, null());
    save_data();
    return ();
}
```

Figure 3.1: The `withdraw_success` message handler
`contracts/SY/evaa/minter.fc#L227-L250`

The `withdraw_success` message handler checks that the message is sent by the base token wallet of the `EvaaSYMinter` contract, but it does not check that the Evaa master contract transferred the base tokens. This allows an attacker to transfer their base tokens with the `withdraw_success` message to burn a lower amount of Evaa SY tokens than the amount being withdrawn by the user, leading to unburned Evaa SY tokens being stuck in the `EvaaSYMinter` contract forever.

Exploit Scenario

Alice transfers 1,000 SY tokens with the `unwrap` message to the `EvaaSYMinter` contract. The Evaa master contract transfers 1,000 base tokens to the `EvaaSYMinter` contract with the `withdraw_success` message in the `forward_payload`. However, Eve sends 500 base tokens to the `EvaaSYMinter` contract with the `withdraw_success` message, which is executed before the Evaa master contract's `withdraw_success` message. Eve's `withdraw_success` message burns 500 Evaa SY tokens and sends her 500 base tokens. However, the burn message sent while processing the Evaa master's `withdraw_success` message fails because of insufficient balance. Alice gets her 1,000 base tokens, but her 500 Evaa SY tokens are stuck in the `EvaaSYMinter` contract forever.

Recommendations

Short term, add a check in the `withdraw_success` message handler to ensure that the Evaa master contract sent the base tokens.

Long term, review all of the transfer notification message handlers to ensure that they include correct access control checks. Consider malicious and fake Jetton transfer messages while implementing access control checks for the transfer notification messages.

4. Users can benefit by sandwiching the index update message to the EvaaSYMinter contract	
Severity: **Low**	Difficulty: **Low**
Type: Timing	Finding ID: TOB-FIVAEVAA-4
Target: `contracts/SY/evaa/minter.fc`	

Description

Users can wrap their underlying token to Evaa SY tokens just before the `storage::index` update and unwrap just after the update to earn four hours of interest in a minute.

The `EvaaSYMinter` contract stores the `storage::index` value to track the interest earned from the Evaa protocol deposits. The FIVA administrator updates the index every four hours to keep it in sync with the Evaa protocol.

When a user wraps their underlying tokens to Evaa SY tokens, the underlying token amount is divided by the `storage::index` value to compute the amount of Evaa SY tokens to mint. Similarly, the Evaa SY token amount is multiplied by the `storage::index` value to compute the amount of underlying tokens to transfer to the user at the time of unwrapping:

```
(int) get_supply_amount(int balance) inline {
    return to_9_decimal(balance) * index_precision / storage::index;
}

(int) get_withdraw_amount(int balance) inline {
    return to_6_decimal(balance) * storage::index / index_precision;
}
```

Figure 4.1: Functions to calculate the wrap and unwrap amounts
`contracts/SY/evaa/getter.fc#L20-L26`

A user can wrap a large amount of underlying tokens just before the `storage::index` update transaction and unwrap just after the index update to earn four hours' interest in a minute at the cost of existing depositors.

Exploit Scenario

The Evaa protocol APY is 8.76%. The FIVA protocol users have wrapped 1,000,000 underlying tokens into Evaa SY tokens. The FIVA administrator will update the index at 4:00 PM from 1 to 1.004. Eve wraps 1,000,000 underlying tokens at 3:59 pm and unwraps the whole amount at 4:01 pm. After unwrapping, Eve gets 1,004,000 underlying tokens back, earning 4,000 tokens in two minutes.

Recommendations

Short term, store the user's last wrap operation timestamp and use it to compute how much interest the user should earn at the time of unwrapping.

Long term, document interest rate update lags in the system's smart contract. Analyze whether attackers can exploit these lags to benefit from the protocol.

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. 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 four issues described in this report, FIVA has resolved three issues and has not resolved the remaining one issue. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Severity	Status
1	Insufficient access control checks in the supply_excess message handler of the EvaaSYMinter contract	High	Resolved
2	Insufficient access control checks in the supply_fail_excess message handler of the EvaaSYMinter contract	High	Resolved
3	Insufficient access control checks in the withdraw_success message handler of the EvaaSYMinter contract	Informational	Resolved
4	Users can benefit by sandwiching the index update message to the EvaaSYMinter contract	Low	Unresolved

Detailed Fix Review Results

TOB-FIVAEVAA-1TOB-FIVA-1: Insufficient access control checks in the supply_excess message handler of the EvaaSYMinter contract

Resolved in PR #115 and PR #171. The `supply_excess` message handler now checks that the message is sent by the Evaa master contract.

TOB-FIVAEVAA-2TOB-FIVA-2: Insufficient access control checks in the supply_fail_excess message handler of the EvaaSYMinter contract

Resolved in PR #115 and PR #171. The `supply_fail_excess` message handler now checks that the `from_addr` is the Evaa master contract.

TOB-FIVAEVAA-3TOB-FIVA-3: Insufficient access control checks in the withdraw_success message handler of the EvaaSYMinter contract

Resolved in PR #115 and PR #171. The `withdraw_success` message handler now checks that the `from_addr` is the Evaa master contract.

TOB-FIVAEVAA-4TOB-FIVA-4: Users can benefit by sandwiching the index update message to the EvaaSYMinter contract

Unresolved.

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

> *We've already aligned with the Evaa protocol team on exposing an onchain getter function for index obtaining. Once this is available, we will implement real-time interest accounting during wrap/unwrap using the most recent index directly from the Evaa protocol.*
>
> *This will eliminate the timing gap between updates and user operations, resolving this class of attacks more effectively.*

C. 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 assessments, supporting client organizations in the technology, defense, blockchain, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, Uniswap, Solana, Ethereum Foundation, Linux Foundation, and Zoom.

To keep up to date with our latest news and announcements, please follow @trailofbits on X or LinkedIn, 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

The sole canonical source for Trail of Bits publications is the Trail of Bits Publications page. Reports accessed through sources other than that page may have been modified and should not be considered authentic.

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be 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 the controls and security properties of software. 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. Their use is also limited by the time and resource constraints of a project.