STON.fi TON AMM DEX V2 Review

Security Assessment

January 9, 2025

Prepared for:
Viacheslav Baranov
STON.fi Holding Ltd.

Prepared by: **Tarun Bansal, Guillermo Larregay, and Elvis Skoždopolj**

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, as well as 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 Twitter 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

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.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Jeff Braswell, Project Manager
jeff.braswell@trailofbits.com

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Tarun Bansal, Consultant
tarun.bansal@trailofbits.com

Guillermo Larregay, Consultant
guillermo.larregay@trailofbits.com

Elvis Skoždopolj, Consultant
elvis.skozdopolj@trailofbits.com

Project Timeline

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

Date	Event
August 13, 2024	Pre-project kickoff call
August 20, 2024	Status update meeting #1
September 05, 2024	Status update meeting #2
September 19, 2024	Delivery of report draft
September 19, 2024	Report readout meeting
October 21, 2024	Delivery of comprehensive report
January 9, 2025	Delivery of updated comprehensive report

Executive Summary

Engagement Overview

STON.fi Holding Ltd engaged Trail of Bits to review the security of STON.fi TON AMM DEX V2. STON.fi supports multiple AMM algorithms such as constant sum, constant product, stable swap, weighted constant product, and weighted stable swap. The decentralized exchange also supports cross-pool and cross-router swaps and liquidity provisions to allow users to execute multiple actions in a single transaction.

A team of three consultants conducted the review from August 13 to September 13, 2024, for a total of eight engineer weeks of effort. Our testing efforts focused on finding issues impacting the availability and integrity of the target system. With full access to source code and documentation, we performed static and dynamic testing of the codebase using automated and manual processes.

Observations and Impact

A total of five issues were found in the STON.fi codebase. Three of these issues (TOB-STONFI-1, TOB-STONFI-2, and TOB-STONFI-4) can potentially affect user funds or internal accounting, while the remaining two issues are low severity and informational.

During the assessment of the codebase maturity, we identified several opportunities to improve the development of STON.fi. In particular, the code would benefit from additional checks in the arithmetic data types and algorithms, log message emission for off-chain monitoring, usage of multisignature wallets for privileged roles, and general improvements in the test suite for edge cases.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that STON.fi Holding Ltd take the following steps:

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

- **Improve the documentation.** Users and developers that integrate with the STON.fi system can benefit from improved and specific documentation regarding the inner workings of the system, its current limitations, and the known issues. Adding more diagrams, explanations, and examples of use increases the understanding of the different parts of the DEX.

- **Improve the test suite.** The implementation of the test suite should not test only the “happy path” but must also consider edge cases, unexpected structures, invalid message fields, and so on. Building a robust test suite increases the likelihood of detecting bugs earlier and, in turn, helps the STON.fi team protect users' funds.

Finding Severities and Categories

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

EXPOSURE ANALYSIS

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

CATEGORY BREAKDOWN

Category	*Count*
Access Controls	1
Data Validation	4

Project Goals

The engagement was scoped to provide a security assessment of the STON.fi TON AMM DEX V2. Specifically, we sought to answer the following non-exhaustive list of questions:

- Are the access controls correctly implemented? Can a non-privileged account gain access to privileged functionality?
- Are messages correctly handled? How are complex structures decoded? Could malformed data be decoded incorrectly, permanently affecting the system?
- Could a user access funds that do not belong to them? Is it possible to drain a contract or otherwise get funds stuck in a contract?
- Can a malicious actor modify internal accounting? Can they affect third parties' interactions with the contracts?
- Are the contract state variables correctly initialized, and their values correctly modified during the contract's lifetime?
- Are the DEX swap and liquidity provider token minting equations correctly implemented? Could malicious actors gain value in a swap or receive more LP tokens than the value added?
- Are sufficient gas checks implemented to avoid an inconsistent system state because of an intermediate message failure?
- Are mathematical function implementations correct? Do they handle all the edge cases to avoid wrong output?

Project Targets

The engagement involved a review and testing of the targets listed below.

TON AMM DEX V2

Repository	https://github.com/ston-fi/dex-core-v2
Version	`commit bb62e6b`
Type	FunC
Platform	TON TVM

TON AMM DEX V2

Repository	https://github.com/ston-fi/dex-core-v2
Version	`commit b2c7e8c` (since August 19)
Type	FunC
Platform	TON TVM

TON AMM DEX V2

Repository	https://github.com/ston-fi/dex-core-v2
Version	`commit 5a529a1` (since September 3)
Type	FunC
Platform	TON 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:

- Manual review of the established user flows for the provided pool implementations:
  - Token swaps
  - Liquidity provision and removal
  - Referral fee accrual and vault withdrawal
- Manual analysis and code review for the mathematical implementations of the pool invariants, including the LP and swap equations and numerical algorithms used for the calculations
- Review and testing of the administration and action messages passed between contracts, their data structures, and how they are handled
- Review of compliance with the Jetton Standard (TEP-74)

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:

- We did not review the `Funcbox` library in depth. Only the functions used by the AMM were reviewed.

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	In several functions, integer parameters lack bound checks to ensure that the value is in the range expected by the function. For example, the square root function from `funcbox` can receive negative argument values that are outside its domain because the argument is a signed integer (TOB-STONFI-3). Some functions, like pool invariants or Jetton output calculations, use fixed-point notation for intermediate results. In many cases, functions take arguments in regular notation and output them in fixed-point notation or vice versa, which can be confusing and prone to errors.	**Moderate**
Auditing	There are no log messages emitted by the system, which makes it difficult to monitor externally. The documentation does not mention an off-chain monitoring system or an incident response plan. We recommend that STON.fi create a document stating what the response will be during an incident. See appendix C for more information.	**Weak**
Authentication / Access Controls	A single account controls the administrator role, and the measures taken to protect its security are not specified. The documentation does not mention the usage of multisignature wallets or hardware keys. See appendix D for guidelines for using multisignature wallets. Messages are authenticated by the sender address, and privileged roles exist for setting configuration parameters, special addresses such as protocol fees controllers, and sender validation for inter-contract messages.	**Moderate**

Complexity Management	In general, the code is clean and easy to read, its complexity is low, and functions scopes are limited. There are some duplicated code blocks in pools.	**Satisfactory**
Decentralization	The router contract is upgradeable by the administrator role. There is a timelock mechanism in place that would allow users to withdraw their assets before the upgrade goes live. System configuration (such as pool fees or parameters) can be changed instantly by the administrator, with no timelock period. Users cannot opt out of these changes. There are privileged actors and accounts in the system.	**Moderate**
Documentation	The documentation provided consists of a high-level description of the different flows using Markdown files with mermaid diagrams. Tests also generate mermaid diagrams for tokens and values sent between contracts, which help developers understand the different messages sent during a DEX operation. There are also references for the different operation codes and the off-chain getters' data structures. The documentation could be improved to have a separate section for end users and developers, including functional and system-level invariants and known risks and limitations.	**Satisfactory**
Low-Level Manipulation	The only low-level code usage in the STON.fi code is in the pool initialization data getters. All other instances of low-level operations (such as FIFT instructions or code) are implemented in the standard library as helpers or functions.	**Strong**
Testing and Verification	The provided test suite runs correctly, and all test cases pass. However, some edge cases, such as issues TOB-STONFI-1 and TOB-STONFI-2, are not covered by the tests. There are no visible integrations of tests with CI/CD	**Moderate**

	pipelines.	
Transaction Ordering	No transaction ordering analysis was performed.	**Further Investigation Required**

Summary of Findings

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

ID	Title	Type	Severity
1	An attacker can steal jettons from a user's lp_account contract	Access Controls	High
2	Protocol fees can be withdrawn by liquidity providers	Data Validation	Medium
3	The math::int::sqrt function returns a result for a negative input	Data Validation	Low
4	Risk of locking jettons in the router contract with cross-router swaps	Data Validation	Medium
5	The convergence threshold in Weighted Stableswap Pool iterations is too large	Data Validation	Informational

Detailed Findings

1. An attacker can steal jettons from a user's lp_account contract

Severity: **High**	Difficulty: **Low**
Type: Access Controls	Finding ID: TOB-STONFI-1
Target: `router/msgs/pool.fc`	

Description

An attacker can send a message with the `cross_provide_lp` operation in the custom payload to steal jettons from a user's `lp_account` contract.

The STON.fi protocol allows users to provide liquidity to a pool by transferring jettons to the `router` contract. Users need to send two jetton transfers: the first jetton transfer creates a `lp_account` contract to store the amount of the jettons transferred, and the second jetton transfer destroys the `lp_account` contract and mints the LP tokens.

The protocol also allows the users to execute a swap and use the swapped jettons to provide liquidity to a pool. Users initiate such a combined action by sending a jetton transfer with the `forward_payload` value containing a swap operation code and a `custom_payload` with the `cross_provide_lp` operation. The jetton transfer notification is received by the `router` contract, which calls the `route_dex_messages` function with the owner of the jetton as the `_caller` argument:

```
(int) handle_jetton_messages() impure inline {
    ...
    if ctx.at(OPCODE) == op::ft::transfer_notification {
        int jetton_amount = in_msg_body~load_coins();
        slice from_address = in_msg_body~load_msg_addr();
        ...
        route_dex_messages(
            ctx.at(SENDER),
            ctx.at(MSG_VALUE),
            jetton_amount,
            from_address,
            in_msg_body~load_ref()
        );
        ...
}
```

Figure 1.1: The jetton transfer notification message handler of the `router` contract (`dex-core-v2/contracts/router/msgs/jetton.fc#L1–L35`)

The `route_dex_messages` function validates the user inputs and sends a message to the `pool` contract:

```
() route_dex_messages(slice _jetton_address, int _available_gas, int _sent_amount,
slice _caller, cell _dex_payload) impure inline {
    ...
    try {
        ...
        reserves::max_balance(storage_fee::router);
        msgs::send_with_stateinit(
            0,
            pool~address(params::workchain),
            pool~state_init(),
            pool::$route(transferred_op, _caller, _sent_amount, _jetton_address,
token_wallet1, _dex_payload),
            CARRY_ALL_BALANCE
        );
    } catch(err_arg, err_code) {
        ...
    }
    return ();
}
```

Figure 1.2: The `route_dex_message` function of the `router` contract
(`dex-core-v2/contracts/router/dex.fc#L84–L91`)

The `pool` contract validates the user inputs and pool reserves, computes the output amount, parses the user-provided `to_address`, and sends a `pay_to` message with the `custom_payload` and `to_address` as the `owner` of the output jettons to the `router` contract to transfer the output jettons to the user:

```
() handle_router_messages() impure inline {
    slice in_msg_body = ctx.at(BODY);

    if ctx.at(OPCODE) == op::swap {
        ...
        slice dex_payload = in_msg_body~load_slice_ref();
        ...
        try { ;; check call_payload body, refund to refund_address if fails
            slice call_payload = dex_payload~load_slice_ref();

            min_out = call_payload~load_coins();
            to_address = call_payload~load_msg_addr();
            fwd_ton_amount = call_payload~load_coins();
            custom_payload_cs = call_payload~load_maybe_ref();
            ...
        } catch(err_arg, err_code) {
            ...
        }

        try {
                ...
                msgs::send_simple(
                    0,
                    ctx.at(SENDER),
                    router::pay_to(
                        to_address,
                        excesses_address,
                        op::swap_ok,
                        fwd_ton_amount,
                        custom_payload_cs,
                        out0,
                        storage::token0_address,
                        out1,
                        storage::token1_address
                    ),
                    CARRY_ALL_BALANCE
                );
                storage::save();
            } catch(err_arg, err_code) {
                ...
            }
            return ();
        }
        ...
}
```

Figure 1.3: The router message handler function of the `pool` contract (`dex-core-v2/contracts/pool/msgs/router.fc#L134–L149`)

The `router` contract checks if the `custom_payload` contains a `cross_swap` or `cross_provide_lp` operation code. If yes, then it calls the `route_dex_messages` function with the `owner` as the `_caller` argument. The `owner` value is parsed from the `pay_to` message body, which was set by the `pool` contract to be the receiver of the output jetton.

However, the `route_dex_messages` function assumes the `owner` to be the initiator of the jetton transfer and sends a message to the `pool` contract to process the liquidity provision operation. A user can specify any address as a receiver of the output jetton for a swap operation. This allows attackers to specify a user address that has a `lp_account` deployed with some non-zero jetton balance and use the victim's balance to mint LP tokens for themselves.

Exploit Scenario

Let us consider a pool of USDC and USDT jettons. Alice transfers 1,000 USDC to the router contract to provide liquidity. The pool contract creates a `lp_account` contract for Alice and the pool and stores the USDC balance of 1,000 tokens. Eve notices this `lp_account` and sends 1,000 USDC to the router contract to swap USDC to USDT and use the received USDT to provide liquidity to the same pool. However, Eve specifies Alice's smart wallet address as the receiver of the swapped USDT tokens. This allows Eve to use the 1,000 USDC balance stored in the `lp_account` created for Alice to provide liquidity to the pool.

Recommendations

Short term, consider one of the following:

1. Propagate the `swap` operation initiator from the pool contract to the `router` contract in the `pay_to` message and use that address as the `_caller` argument when calling the `route_dex_messages` function.
2. Do not allow users to specify a different `to_address` for the `swap` if the `custom_payload` contains the `cross_provide_lp` operation code.

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.

2. Protocol fees can be withdrawn by liquidity providers

Severity: **Medium**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-STONFI-2
Target: `pool/msgs/lp_account.fc`	

Description

Liquidity providers can withdraw the protocol fees accumulated when providing unbalanced liquidity since they are never removed from the pool reserves.

When a user provides liquidity, a message is sent from the `lp_account` contract to the `pool` contract with the `op::cb_add_liquidity` opcode. If the pool already has some liquidity, the `pool::get_lp_provide_out` function will be called to calculate the liquidity that needs to be minted and the protocol fee for each jetton, as shown in figure 2.1:

```
} else {
    int new_collected_protocol_fees0 = 0;
    int new_collected_protocol_fees1 = 0;

    try { ;; catch math errors
        (liquidity, new_collected_protocol_fees0, new_collected_protocol_fees1) =
pool::get_lp_provide_out(tot_am0, tot_am1);
    } catch(_, _) { }

    storage::total_supply_lp += liquidity;
    ;; one side will be 0
    storage::collected_token0_protocol_fee += new_collected_protocol_fees0;
    storage::collected_token1_protocol_fee += new_collected_protocol_fees1;
}

storage::reserve0 += tot_am0;
storage::reserve1 += tot_am1;
```

Figure 2.1: Adding liquidity to a non-empty pool (`dex-core-v2/contracts/pool/msgs/lp_account.fc#L44–L59`)

However, the protocol fee is never removed from the pool jetton reserves. As a result, the fee can be withdrawn by the liquidity providers when burning their liquidity. Since the `collected_token0_protocol_fee` and `collected_token1_protocol_fee` storage variables will still be non-zero, the protocol fee can still be withdrawn by the `protocol_fee_address`. This can lead to another pool's jetton balance being lower than the pool's reserves.

Exploit Scenario
Two constant product pools are deployed, both using USDT as one of the jettons. Eve deposits unbalanced liquidity to the second pool, causing a protocol fee to be accrued for USDT. Eve withdraws her liquidity of the second pool, and the `protocol_fee_address` withdraws the accrued protocol fees. The liquidity of the first pool is now lower than the reserves of this pool, causing a loss for the first pool's liquidity providers.

Recommendations
Short term, remove the protocol fees from the jetton reserves of a pool when unbalanced liquidity is provided.

Long term, improve the testing suite by ensuring that all expected side-effects of each action are verified.

3. The math::int::sqrt function returns a result for a negative input	
Severity: **Low**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-STONFI-3
Target: `funcbox/contracts/math/int/int.fc`	

Description

The `math::int::sqrt` function does not validate that the input parameter is a non-negative number.

If the input parameter for the `math::int::sqrt` function is a negative number, this function will either throw an error (in rare cases) or return an incorrect negative value as the result. The function is shown in figure 3.1:

```
int math::int::sqrt(int x) inline {
    if (x == 0) { return x; }

    int r = 181;
    int xx = x;
    if xx >= 0x100000000000000000000000000000000 {
        xx >>= 128;
        r <<= 64;
    }
    if xx >= 0x10000000000000000 {
        xx >>= 64;
        r <<= 32;
    }
    if xx >= 0x100000000 {
        xx >>= 32;
        r <<= 16;
    }
    if xx >= 0x10000 {
        xx >>= 16;
        r <<= 8;
    }

    r = (r * (xx + 65536)) >> 18;

    repeat(7) {
        r = (r + x / r) >> 1;
    }

    int r1 = x / r;

    return (r < r1 ? r : r1);
}
```

Figure 3.1: The `math::int::sqrt` function
(`funcbox/contracts/math/int/int.fc#L7–L38`)

Developers who use this function could incorrectly assume that the function will throw an error or return the negative of the square root of the absolute value of the input, which can lead to vulnerabilities being introduced in the integrating contracts.

Exploit Scenario

An AMM uses the `math::int::sqrt` function to determine the amount of liquidity to be minted for a constant product liquidity pool. Expecting negative numbers to cause the function to throw an error, they forget to validate the input, causing a bug in the AMM.

Recommendations

Short term, add validation to the `math::int::sqrt` function that ensures that the input is a non-negative number.

Long term, improve the testing suite of the `funcbox` repository by adding additional unit tests and by using advanced testing techniques such as stateless fuzzing to further verify the properties of the provided functions. This is especially useful for mathematical functions where strict mathematical properties can be easily applied. Consider implementing differential fuzzing of mathematical functions against a reference implementation.

4. Risk of locking jettons in the router contract with cross-router swaps

Severity: **Medium**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-STONFI-4
Target: `router/msgs/pool.fc`	

Description

A cross-router swap operation can result in the middle jettons getting stuck in the second router because of insufficient gas checks.

The `pay_to` message handler of the `router` contract sets the `fwd_ton_amount` value to zero in case the available gas is less than the user-provided value of the `fwd_ton_amount`:

```
if ctx.at(OPCODE) == op::pay_to {
    ...
    int used_gas = ctx.at(FWD_FEE) + gas::router::pay_to;
    if (((fwd_opcode == op::cross_swap) | (fwd_opcode == op::cross_provide_lp)) &
        ...
    } else {
        ;; send jettons normally

        ;; check gas
        int gas_available = ctx.at(MSG_VALUE) - (used_gas + storage_fee::router); ;;
estimate free gas after this call
        if (fwd_ton_amount > 0) & (
            (gas_available - (gas::jetton_wallet::est_transfer + (ctx.at(FWD_FEE) *
2))) - fwd_ton_amount <= 0
        ) {
            ;; do not send transfer_notification, if gas is too low
            fwd_ton_amount = 0;
        }

        reserves::max_balance(storage_fee::router);
        ;; one is always zero
        if amount0_out > 0 {
            msgs::send_simple(
                0,
                token0_address,
                jetton_wallet::transfer(fwd_ton_amount, amount0_out, owner,
excesses_address)
                    .store_maybe_ref(custom_payload)
                    .end_cell(),
                QCARRY_ALL_BALANCE
            );
        } else {
            msgs::send_simple(
                0,
                token1_address,
                jetton_wallet::transfer(fwd_ton_amount, amount1_out, owner,
excesses_address)
                    .store_maybe_ref(custom_payload)
                    .end_cell(),
                QCARRY_ALL_BALANCE
            );
        }
    }

    return (true);
}
```

Figure 4.1: The `pay_to` message handler of the `router` contract
(`dex-core-v2/contracts/router/msgs/pool.fc#L58–L65`)

This gas check has been implemented to ensure a successful jetton transfer to the receiver. However, setting the `fwd_ton_amount` to zero prevents the receiver's jetton wallet contract from sending a `transfer_notification` message to the receiver. Without the transfer notification, in cases where the receiver is a smart contract that does not allow anyone to withdraw its jetton balance, the jettons get stuck in the receiver contract forever.

Specifically, the cross-router swap relies on transferring a jetton from one router contract to another, and in the absence of the `transfer_notification` message, the jettons get stuck in the second router contract forever.

Exploit Scenario

Let us consider two routers: router1 with token1 and token2 and router2 with token2 and token3. Alice transfers her token1 jettons to router1 with a `forward_payload` containing a cross-router swap payload, which includes two swap payloads; the second payload is added as a `custom_payload` in the first payload. By mistake, she specifies the same value of the `fwd_ton_amount` for both of the swap payloads. The router1 executes the swap and transfers the token2 to router2, but does not send a `transfer_notification` message to the router2 contract. This results in the token2 getting stuck in router2.

Recommendations

Short term, consider one of the following:

1. In case there is insufficient gas to send as `fwd_ton_amunt`, transfer the output jettons to the original caller, excess address, or refund address to ensure that they are not stuck in a contract forever.

2. Parse the `custom_payload` of a `swap_payload` to get the value of the specified `fwd_ton_amount` and ensure that the user-provided gas is enough to send the required `fwd_ton_amount` before processing the swap message. Note that the

router contract needs to parse all of the `custom_payload` reference cells recursively to parse all the `fwd_ton_amount` values to implement the gas check.

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

5. The convergence threshold in Weighted Stableswap Pool iterations is too large

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-STONFI-5
Target: `pool/pools/weighted_stableswap/math.fc`	

Description

The weighted stableswap pool calculates the swap outcome using an iterative algorithm to find the output jetton amount that makes the invariant after the swap equal to the invariant before the swap. As is the case with most iterative algorithms used to solve problems in numerical calculus, a threshold value is set as a convergence check to stop running the iterations.

In the particular case of the implementation of this pool, the value for the threshold (the constant `__EPSILON`) is too large, as shown in figure 5.1

```
const int __EPSILON = 1000000000000000000000000000;
const int __MAX_ITERATIONS = 255;

[...]

(int) solve_dx(int x, int y, int dy) impure inline {
    int p = storage::w0;
    int q = storage::w0.math::fp::complement();

    int start_x = x;
    int k = _invariant(x, y, p, q);
    int i = 0;
    int delta_f = 0;

    do {
        delta_f = _invariant(start_x, y + dy, p, q) - k;
        int df_d = _dx_invariant(start_x, y + dy, p, q);

        throw_if(error::df_zero, df_d == 0);
        start_x -= math::fp::to(delta_f.math::fp::div(df_d));

        i += 1;
        throw_if(error::not_converge, i > __MAX_ITERATIONS);
    } until(abs(delta_f) <= __EPSILON);

    return (x - start_x);
}
```

```
(int) solve_dy(int x, int y, int dx) impure inline {
    int p = storage::w0;
    int q = storage::w0.math::fp::complement();

    int start_y = y;
    int k = _invariant(x, y, p, q);
    int i = 0;
    int delta_f = 0;

    do {
        delta_f = _invariant(x + dx, start_y, p, q) - k;
        int df_d = _dy_invariant(x + dx, start_y, p, q);

        throw_if(error::df_zero, df_d == 0);
        start_y -= math::fp::to(math::fp::div(delta_f, df_d));

        i += 1;
        throw_if(error::not_converge, i > __MAX_ITERATIONS);
    } until(abs(delta_f) <= __EPSILON);

    return (y - start_y);
}
```

Figure 5.1: The __EPSILON threshold
(`dex-core-v2/contracts/pool/pools/weighted_stableswap/math.fc`)

In the pool's implementation, the value equals 10^{27} and corresponds to 1 TON expressed in 18-decimal fixed-point notation. This can cause the invariant to be off by at most 1 TON, which can skew the calculation and result in an incorrect number of output tokens depending on the pool's current balances.

Recommendations

Short term, reduce the threshold to ensure that the calculation is more accurate, even if the number of iterations increases. This will be more gas costly, but swaps will be fairer for users.

Long term, analyze the algorithm's convergence speed and precision in terms of the number of correct decimals gained per iteration and establish an optimal threshold value. This optimization will need to trade off the implementation's gas costs and the maximum value expected to be lost by users during swaps.

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 is not applicable 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. Incident Response Recommendations

This section provides recommendations on formulating an incident response plan.

- **Identify the parties (either specific people or roles) responsible for implementing the mitigations when an issue occurs (e.g., deploying smart contracts, pausing contracts, upgrading the front end, etc.).**
- **Document internal processes for addressing situations in which a deployed remedy does not work or introduces a new bug.**
    - Consider documenting a plan of action for handling failed remediations.
- **Clearly describe the intended contract deployment process.**
- **Outline the circumstances under which STON.fi Holding Ltd will compensate users affected by an issue (if any).**
    - Issues that warrant compensation could include an individual or aggregate loss or a loss resulting from user error, a contract flaw, or a third-party contract flaw.
- **Document how the team plans to stay up to date on new issues that could affect the system; awareness of such issues will inform future development work and help the team secure the deployment toolchain and the external on-chain and off-chain services that the system relies on.**
    - Identify sources of vulnerability news for each language and component used in the system, and subscribe to updates from each source. Consider creating a private Discord channel in which a bot will post the latest vulnerability news; this will provide the team with a way to track all updates in one place. Lastly, consider assigning certain team members to track news about vulnerabilities in specific system components.
- **Determine when the team will seek assistance from external parties (e.g., auditors, affected users, other protocol developers) and how it will onboard them.**
    - Effective remediation of certain issues may require collaboration with external parties.
- **Define contract behavior that would be considered abnormal by off-chain monitoring solutions.**

It is best practice to perform periodic dry runs of scenarios outlined in the incident response plan to find omissions and opportunities for improvement and to develop “muscle memory.” Additionally, document the frequency with which the team should perform dry runs of various scenarios, and perform dry runs of more likely scenarios more regularly. Create a template to be filled out with descriptions of any necessary improvements after each dry run.

D. Security Best Practices for Using Multisignature Wallets

Consensus requirements for sensitive actions, such as spending funds from a wallet, are meant to mitigate the risks of the following:

- Any one person overruling the judgment of others
- Failures caused by any one person's mistake
- Failures caused by the compromise of any one person's credentials

For example, in a 2-of-3 multisignature wallet, the authority to execute a "spend" transaction would require a consensus of two individuals in possession of two of the wallet's three private keys. For this model to be useful, the following conditions are required:

1. The private keys must be stored or held separately, and access to each one must be limited to a unique individual.
2. If the keys are physically held by third-party custodians (e.g., a bank), multiple keys should not be stored with the same custodian. (Doing so would violate requirement #1.)
3. The person asked to provide the second and final signature on a transaction (i.e., the cosigner) should refer to a pre-established policy specifying the conditions for approving the transaction by signing it with his or her key.
4. The cosigner should also verify that the half-signed transaction was generated willingly by the intended holder of the first signature's key.

Requirement #3 prevents the cosigner from becoming merely a "deputy" acting on behalf of the first signer (forfeiting the decision-making responsibility to the first signer and defeating the security model). If the cosigner can refuse to approve the transaction for any reason, the due-diligence conditions for approval may be unclear. That is why a policy for validating transactions is needed. A verification policy could include the following:

- A protocol for handling a request to cosign a transaction (e.g., a half-signed transaction will be accepted only via an approved channel)
- An allowlist of specific addresses allowed to be the payee of a transaction
- A limit on the amount of funds spent in a single transaction or in a single day

Requirement #4 mitigates the risks associated with a single stolen key. For example, say that an attacker somehow acquired the unlocked Ledger Nano S of one of the signatories. A voice call from the cosigner to the initiating signatory to confirm the transaction would reveal that the key had been stolen and that the transaction should not be cosigned. If the signatory were under an active threat of violence, he or she could use a duress code (a code word, a phrase, or another signal agreed upon in advance) to covertly alert the others that the transaction had not been initiated willingly, without alerting the attacker.

E. 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 a comprehensive analysis of the system.

From September 25 to September 27, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the STON.fi 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 five issues described in this report, STON.fi has resolved four issues and has not resolved the remaining one issue. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	An attacker can steal jettons from a user's lp_account contract	Resolved
2	Protocol fees can be withdrawn by liquidity providers	Resolved
3	The math::int::sqrt function returns a result for a negative input	Resolved
4	Risk of locking jettons in the router contract with cross-router swaps	Unresolved
5	The convergence threshold in Weighted Stableswap Pool iterations is too large	Resolved

Detailed Fix Review Results

TOB-STONFI-1: An attacker can steal jettons from a user's lp_account contract
Resolved in PR 1. The `pay_to` message now includes an `original_caller` field that is the value of the sender of the first jetton transfer. This `original_caller` value is now used as the `_caller` argument of the `route_dex_messages` function to resolve the issue. Additionally, the `op::cross_provide_lp` has been removed from the protocol.

TOB-STONFI-2: Protocol fees can be withdrawn by liquidity providers
Resolved in PR 1. The `op::cb_add_liquidity` operation handler of the `pool` contract now subtracts the protocol fee from the token amounts before adding them to the pool reserves.

TOB-STONFI-3: The math::int::sqrt function returns a result for a negative input
Resolved in commit 95a60be of the `funcbox` repository. The `math::int::sqrt` function now throws an exception for negative numbers. The `package.json` file of the STON.fi project has been updated to include the fixed version of the FuncBox package in PR 1.

Additionally, a change was introduced in commit 77af753 where the `funcbox` implementation of the square root was changed and moved to `pools/constant_product/math.fc`. The new square root calculation algorithm is based on OpenZeppelin contracts v5.1.0, and ported to FunC. There are two minor differences between OZ and STON.fi implementations:

- The OZ version calculates Newton's method iterations six times, while STON.fi iterates seven times, which is consistent with older OZ versions. This should be fine, as it could give precision identical to or potentially more than OZ, at the expense of gas for an extra iteration.
- The STON.fi version calculates an intermediate `r1` value that should be equal to `r`, given that `r == sqrt(x)`. If there is any difference, the smallest amount is returned. The difference should be in the order of 1 LSB or less.

TOB-STONFI-4: Risk of locking jettons in the router contract with cross-router swaps
Unresolved. The issue has not been resolved.

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

> *We have decided not to address this issue as it was deemed irrelevant following our internal review. Setting `fwd_amount` to 0 is necessary to prevent jettons from becoming stuck in the router due to failed gas checks. Potential fixes could introduce unpredictable behavior in the contract.*

TOB-STONFI-5: The convergence threshold in Weighted Stableswap Pool iterations is too large

Resolved in PR 1. The value of the `__EPSILON` has been changed to `1000000000000000000000000`, which is equal to `0.001` TON expressed in 18 decimal fixed-point notation.

F. 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.