TON Foundation Stablecoin

Security Assessment

March 25, 2024

Prepared for:
Dr. Ilya Elias
TON Foundation

Prepared by: **Guillermo Larregay and Tarun Bansal**

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.
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
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:

Anne Marie Barry, Project Manager
annemarie.barry@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:

Guillermo Larregay, Consultant
guillermo.larregay@trailofbits.com

Tarun Bansal, Consultant
tarun.bansal@trailofbits.com

Project Timeline

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

Date	Event
February 12, 2024	Pre-project kickoff call
February 20, 2024	Status update meeting #1
February 27, 2024	Delivery of report draft
February 27, 2024	Report readout meeting
March 25, 2024	Delivery of comprehensive report

Executive Summary

Engagement Overview

TON Foundation engaged Trail of Bits to review the security of Stablecoin contracts. The provided contracts are based on the TEP-74 Fungible Tokens (Jettons) standard for TON Blockchain, and add a new layer of privileges that allow centralized institutions to have complete control over minting, burning, and transfers of the Stablecoin Jettons.

A team of two consultants conducted the review from February 12 to February 23, 2024, for a total of four 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, as well as the provided test suite, we performed static and dynamic testing of the Stablecoin contract repository using automated and manual processes.

Observations and Impact

The full system review unveiled one high-severity issue and one informational issue. In particular, TOB-TON-STABLE-1 can affect the issuer's ability to control funds and potentially the whole system, and TOB-TON-STABLE-2 relates to an unneeded bounceable flag in a message.

The Codebase Maturity evaluation shows that there is still some work to be done to increase the auditability of the system, to improve the documentation and test suite, and to minimize the usage of low-level instructions and structures in the code.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that TON Foundation take the following steps prior to the main network launch of the Stablecoin contracts:

- **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 documentation coverage.** The current state of the codebase requires the stablecoin issuer to read through different sources of documentation to gather all the required information to understand or improve the system.

- **Improve the test suite.** In the current form, the test suite has only unit tests. To ensure better coverage and the early detection of issues such as TOB-TON-STABLE-1, the test suite should also include integration tests, and all of the test cases should be performed with either real data, or mocks that try to replicate real data.

Finding Severities and Categories

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

EXPOSURE ANALYSIS

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

CATEGORY BREAKDOWN

Category	*Count*
Data Validation	1
Undefined Behavior	1

Project Goals

The engagement was scoped to provide a security assessment of the TON Foundation Stablecoin contracts. Specifically, we sought to answer the following non-exhaustive list of questions:

- Can users lose their funds or access to their wallets due to their own mistakes or incorrect operations?
- Can the issuer lose access to the minter contract due to their own mistakes or incorrect operations?
- Can users prevent the issuer from locking the funds in their wallets? Can users recover funds from locked wallets?
- Is it possible for an upgrade operation to lock either the minter or wallets contracts? Are system invariants, such as total amount of minted tokens or administrator address, preserved after the upgrade?
- Are all privileged functions working correctly? Are the mechanisms that prevent unauthorized users working correctly?
- Does the token adhere to the TEP-74 standard?
- Are all inputs validated?

Project Targets

The engagement involved a review and testing of the following target.

stablecoin-contract

Repository	https://github.com/ton-blockchain/stablecoin-contract
Version	`baa714c50d75859bc24dd52f35c4d5b34337bb08`
Type	FunC / TL-B
Platform	TON Blockchain

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 provided codebase.** The Stablecoin codebase implements a standard fungible token that can be fully controlled by a central issuing entity, and has complete control over the minting and burning of the tokens and the user balances. The system is composed primarily of two parts: the Jetton Minter and the Jetton Wallet contracts, both of which can be upgraded by the issuer. We reviewed the mentioned contracts and the related libraries and utility files for common FunC vulnerabilities, such as not marking functions as impure, calling modifying functions with the non-modifying syntax, not handling bounced messages, and not forwarding sufficient gas with the messages. We also reviewed the codebase for compliance with the TEP-74 standard and fulfillment of the requirements specified by the TON Foundation.

- **Analysis of the upgrade mechanism.** We analyzed the upgrade mechanisms in the minter and wallet contracts for potential errors, including: irreversible upgrades, loss of funds or control over funds, wrong or unverified data after the upgrades, and other issues that could lead to value loss for the issuer.

- **Review of the test suite.** The test suite was used as a reference for use cases and user flows, and we considered the provided test cases for issues such as data validation errors, incorrect parameters to functions, and issues with async function calls, among others.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- We did not consider the security implications of the administrator account. The TON Foundation team stated that this should be a multisignature wallet with proper key handling.

- We had no access to the off-chain message creation scripts or interfaces. As a result, we were unable to review the correctness of the message bodies generated by these interfaces.

- The test suite provided mostly deals with mock data that sometimes is not similar to real-life scenarios. We considered this a limitation of the test suite, and did not modify the tests in any way unless we had to test a specific situation such as TOB-TON-STABLE-1.

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	Arithmetic is used mostly to keep track of Jetton balances. The calculations are simple additions and subtractions, and there are no rounding issues in these cases. Integer types are used in the code.	**Strong**
Auditing	No logging messages are emitted for critical operations or parameter changes, such as administrator changes in the Minter contract or status changes in the Wallet contracts. All state-modifying operations should be easily traceable from outside the blockchain, in order for early incident response alerts to be triggered. Since this is a template contract that can be used and modified by third parties, it is understandable that there are no related TON Foundation incident response plans. However, we recommended that the base contracts implement features that facilitate easy off-chain monitoring and provide a basic template for third parties to create their own incident response plans.	**Weak**
Authentication / Access Controls	All privileged functions from the Minter contract are managed by a single administrator address. However, the README file clarifies that this administrator should be a multi-signature wallet with proper key management procedures. The administrator address change is done via a two-step procedure. The Wallet contract has an administrative role that is associated with the Minter, and a less-privileged role for the user that owns the wallet. The Minter can burn and lock funds in the wallets, and the user can transfer their Jettons to other wallets.	**Satisfactory**
Complexity	In general, the functions are short, concise, and easy to	**Satisfactory**

Management	read. There is no code duplication in the codebase, since many functions rely on the FunC standard library. Data validation is performed for every message field, but the validation is done in two parts: first, the message headers and operation code are checked, the correct operation handler is called, and then each handler validates the actual message content.	
Decentralization	The system is designed to be fully centralized. The stablecoin issuers should have control over minting, burning, and user funds. Issuers can also limit the extent of users' interaction with the Jetton wallets.	**Weak**
Documentation	There is no specific documentation regarding the differences between a standard Jetton and the Stablecoins contract, other than the README file. In general, the code comments are sparse, and usually indicate message or storage layouts. New features, such as the usage of TVM Version 6 (not yet live on mainnet) instructions for gas calculations, are not sufficiently documented in the codebase.	**Weak**
Low-Level Manipulation	There are no instances of low-level interaction or assembly language in the Minter and Wallet contracts, but they rely on the FunC standard library that implements low-level calls. Some of the features and instructions used are not yet live on TON main network, but are expected to go live before the release of the Stablecoin contracts.	**Moderate**
Testing and Verification	The provided unit tests are designed to ensure that the features work, but they do not reflect the potentially adversarial nature of real-world scenarios. There are no integration tests for user or administrative flows. For example, issue TOB-TON-STABLE-1 could have been prevented with an end-to-end integration test, using a new version of the wallet contract instead of a mock cell.	**Weak**
Transaction Ordering	The TON Blockchain is asynchronous by design, and as such, some transaction ordering processes can be considered risky.	**Satisfactory**

	The Stablecoin contracts do not implement any specific measure to prevent these kinds of attacks, they rely on the network's prevention measures. The documentation contains no mentions of risks. In general, the exposure to transaction ordering attacks is small and limited in nature.	

Summary of Findings

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

ID	Title	Type	Severity
1	Upgrade operation can brick the minter and wallets	Data Validation	**High**
2	The provide_wallet_address operations sends a bounceable message as the response	Undefined Behavior	**Informational**

Detailed Findings

1. Upgrade operation can brick the minter and wallets	
Severity: **High**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-TON-STABLE-1
Target: `stablecoin-contract/contracts/jetton-minter.fc`	

Description

The `upgrade` operation, available to the administrator account of the Jetton minter contract, does not perform any data validation in the cells received as parameters, and upgrades both the code and data of the minter. There is no option to upgrade one or the other independently.

```
() recv_internal(int msg_value, cell in_msg_full, slice in_msg_body) impure {
    [...]
    if (op == op::upgrade) {
        throw_unless(error::not_owner, equal_slices_bits(sender_address,
admin_address));
        (cell new_data, cell new_code) = (in_msg_body~load_ref(),
in_msg_body~load_ref());
        in_msg_body.end_parse();
        set_data(new_data);
        set_code(new_code);
        return ();
    }
    [...]
}
```

Figure 1.1: The `op::upgrade` operation in Jetton minter's `recv_internal` function (`stablecoin-contract/contracts/jetton-minter.fc#L58–L241`)

This approach may render the Jetton minter and wallet contracts unusable for the following reasons:

- Setting an incorrect `total_supply` value in the data cell can break the internal accounting of circulating coins.
- Setting an invalid `admin_address` results in loss of access to the admin functions.
- Upgrading the `jetton_wallet_code` breaks the system:

    - The Minter contract can no longer access wallets created with the old wallet code, since the address is calculated using the `jetton_wallet_code` information.
    - Old and new Jetton wallets cannot interact with each other for the same reason.
    - Most importantly, the administrator account will lose access to locking, unlocking, and managing funds of the existing wallets with old code, leading to users losing funds and administrators losing control over tokens in old wallets.
- The same can happen with the new Jetton minter code cell if the upgraded version lacks critical features.

The administrator account is supposed to be a multi-signature wallet requiring a threshold of signers to agree on the update. However, in the current state of the upgrade system, the signers need to check for data validity manually, which imposes an extra burden.

Exploit Scenario

An upgrade to the minter contract accidentally sets the `admin_address` field of the data cell to zero, making further upgrades or execution of privileged functions impossible. This can lead to funds or value loss for the stablecoin users and the company issuing the coin.

Alternatively, a developer adds new interesting features to the Jetton wallet contract, and the stablecoin issuer decides to upgrade their minter contract to use the new features. The system is no longer usable by the existing users, and the administrators lose access to the old wallets.

Recommendations

Short term, remove the ability to upgrade the data cell. There are already functions to upgrade parts of the data cell, such as the two-step administrator address change or the `change_metadata_uri` operation. If there is a need to manually change the `total_supply` field, a new function and operation should be added with that specific goal. Finally, the `jetton_wallet_code` should not be upgraded.

Long term, reconsider the upgradeability of the Jetton wallet code, and if needed, devise a new upgrade mechanism that does not break interaction with the existing wallets. Document the risks in the upgrade procedure and provide stablecoin issuers a way to test the upgrades before they go live in mainnet. Ensure that the multi-signature wallet signers are aware of the upgrade and its potential consequences.

2. The provide_wallet_address operation sends a bounceable message as the response

Severity: **Informational**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-TON-STABLE-2
Target: `stablecoin-contract/contracts/jetton-minter.fc`	

Description

The `provide_wallet_address` operation handler sends a bounceable message as the response, but the `recv_internal` function does not handle it as a bounced message.

The `provide_wallet_address` operation is implemented to allow other smart contracts to query the Jetton wallet address of an owner address. The code block handling the `provide_wallet_address` operation calculates the wallet address of the provided `owner_address` and sends a bounceable message to the `sender_address` with the `take_wallet_address` operation:

```
if (op == op::provide_wallet_address) {
    ;; see provide_wallet_address TL-B layout in jetton.tlb
    slice owner_address = in_msg_body~load_msg_addr();
    int include_address? = in_msg_body~load_bool();
    in_msg_body.end_parse();

    cell included_address = include_address?
    ? begin_cell().store_slice(owner_address).end_cell()
    : null();

    ;; build MessageRelaxed, see TL-B layout in stdlib.fc#L733
    var msg = begin_cell()
    .store_msg_flags_and_address_none(BOUNCEABLE)
    .store_slice(sender_address)
    .store_coins(0)
    .store_prefix_only_body()
    .store_op(op::take_wallet_address)
    .store_query_id(query_id);

    if (is_same_workchain(owner_address)) {
        msg = msg.store_slice(calculate_user_jetton_wallet_address(owner_address,
my_address(), jetton_wallet_code));
    } else {
        msg = msg.store_address_none();
    }

    cell msg_cell = msg.store_maybe_ref(included_address).end_cell();
    send_raw_message(msg_cell, SEND_MODE_CARRY_ALL_REMAINING_MESSAGE_VALUE |
SEND_MODE_BOUNCE_ON_ACTION_FAIL);
    return ();
}
```

Figure 2.1: The `op::provide_wallet_address` operation in the Jetton minter's `recv_internal` function (`stablecoin-contract/contracts/jetton-minter.fc#L133–L162`)

However, the code block of the `recv_internal` function that handles bounced messages ignores the messages with operation `provide_wallet_address`. Therefore, there is no benefit of sending a bounceable message because no state update needs to be reverted in case the message is bounced from the destination address.

Additionally, the sender of the `provide_wallet_address` message will send Ton to pay for gas with the message and will need to receive back the remaining amount. However, if the `take_wallet_address` operation message bounces, it will carry the remaining gas back to the Jetton minter contract, causing a loss for the original sender.

Recommendations

Short term, send an unbounceable message as the response from the `provide_wallet_address` message handler.

Long term, use bounceable messages to revert a state change in case of a failure on the destination address and ensure that all bounced messages are handled correctly.

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. Code Quality Recommendations

This appendix contains findings that do not have immediate or obvious security implications. However, they may facilitate exploit chains targeting other vulnerabilities, become easily exploitable in future releases, or decrease code readability. We recommend fixing the issues reported here.

- **Replace magic integers in code with named constants.** In the `jetton-wallet.fc` file, the different locking statuses are identified by integer constants in code. For example, to check if the wallet is not able to receive Jettons, the check performed is `(status & 2)`. In that case, the 2 should be replaced with a named constant to improve code readability and avoid coding mistakes if new status values are included in the future.

- **Document the decision of making Stablecoin Jettons messages compatible with Standard Jettons.** The prefixes used for messages in `jetton.tlb` are the same as the ones for TEP-74 Jettons, even if the data types are not exactly the same, but compatible. This design decision should be explicitly documented.

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 a comprehensive analysis of the system.

On March 29, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the TON Foundation 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 two issues described in this report, TON Foundation has resolved one issue and has not resolved the remaining one issue. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Upgrade operation can brick the minter and wallets	**Unresolved**
2	The provide_wallet_address operation sends a bounceable message as the response	**Resolved**

Detailed Fix Review Results

TOB-TON-STABLE-1: Upgrade operation can brick the minter and wallets

Unresolved. Changes implemented in commit 9a53e28. The TON Foundation team updated the `README.md` to warn the user to check the upgrade functionality on the testnet before executing it on the mainnet.

The changes provided in the mentioned commit do not address the root cause of the issue, as it only updates the documentation. As a consequence, it is still possible to brick the contracts with an untested upgrade, and jetton-wallet code upgrades still make them incompatible with any old deployed versions.

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

The upgrade procedure can be fully checked in the testnet and then verify that the upgrade message is identical to what we have checked. We have added information about the need for this in readme. The alternative would be to put low-level checks on TON assembler into the contract, which is difficult to audit. Such checks are made in a separate branch upgrade_extra_checks in emulation-utils.fc but we collectively decided not to merge them.

TOB-TON-STABLE-2: The provide_wallet_address operation sends a bounceable message as the response

Resolved in commit 9a53e28. The `provide_wallet_address` operation handler now sends an unbounceable message as the response to the caller.

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.