BEOSIN
Blockchain Security

# Aqua Protocol

Smart Contract Security Audit

No. 202407221416

July 22th, 2024

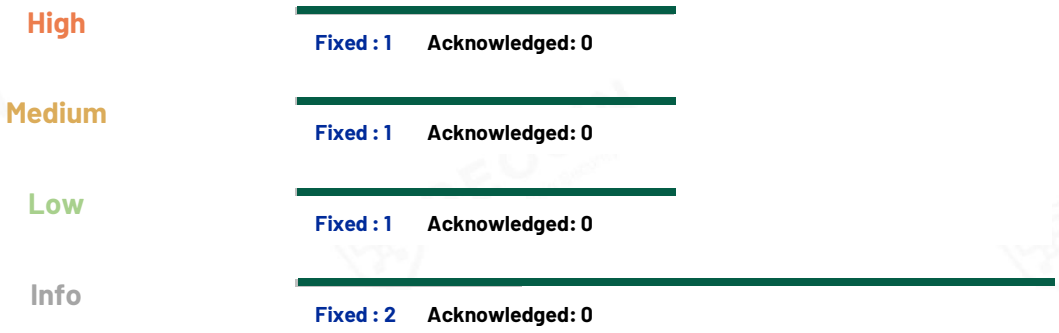
SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

# Contents

# Summary of Audit Results

After auditing, 1 High risk, 1 Medium risk, 1 Low risk and 2 Info item was identified in the Aqua Protocol project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:




- **Risk Description:**

1. In Aqua Protocol project, owner are requested to keep their private keys safe and avoid leakage of private keys to avoid hacker attacks.

- **Project Description:**

The Aqua Protocol is a decentralized lending platform based on the TON blockchain that allows users to borrow stablecoin AquaUSD by staking TON coins or Liquid Staked Tokens (LSTs). The following is an introduction to the main functions of the project:

➢ Mint function explained:

Alice first initiates a transfer to Master's Jetton wallet through her own Jetton wallet, accompanying it with a minting request. Upon receiving the transfer and minting request from Alice, Master's Jetton wallet forwards the minting request to the Aqua-Master contract for processing. Upon receiving the request, the Aqua-Master contract executes an internal transfer process, directly minting the corresponding amount of Aqua tokens into Alice's Aqua wallet.

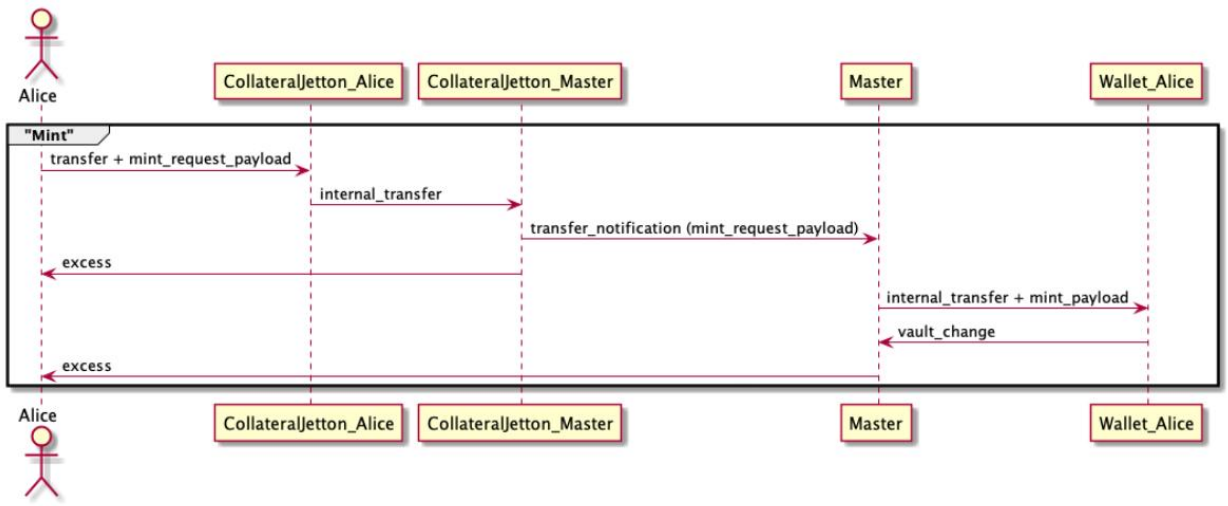



Figure 1 Schematic diagram of the minting process

➢ Withdraw function explained:

Firstly, Alice initiates a withdrawal request by invoking the Aqua-Master contract. Upon receiving and processing Alice's withdrawal request, the Aqua-Master contract sends a processing instruction to Alice's Aqua-wallet. Once the Aqua-wallet receives this instruction, it will first settle Alice's borrowing fees, then deduct the amount of collateral Alice wishes to withdraw according to the withdrawal request. After completing these operations, the Aqua-wallet conducts a collateral ratio check to ensure that Alice's collateral ratio remains at a healthy level. If Alice's collateral ratio meets the health standard, the Aqua-wallet will send the results of the entire process, including deduction details, collateral ratio status, etc., back to the Aqua-Master contract in the form of a message. Upon receiving the confirmation message from the Aqua-wallet, the Aqua-Master contract will further notify Master's Jetton wallet to proceed with the transfer operation, transferring the assets Alice has withdrawn from

Master's Jetton wallet to Alice's Jetton wallet. After the transfer is completed, the Aqua-Master contract updates a series of key data, including but not limited to: total_supply, asset_total_supply, total_collateral_locked, and fee_total.

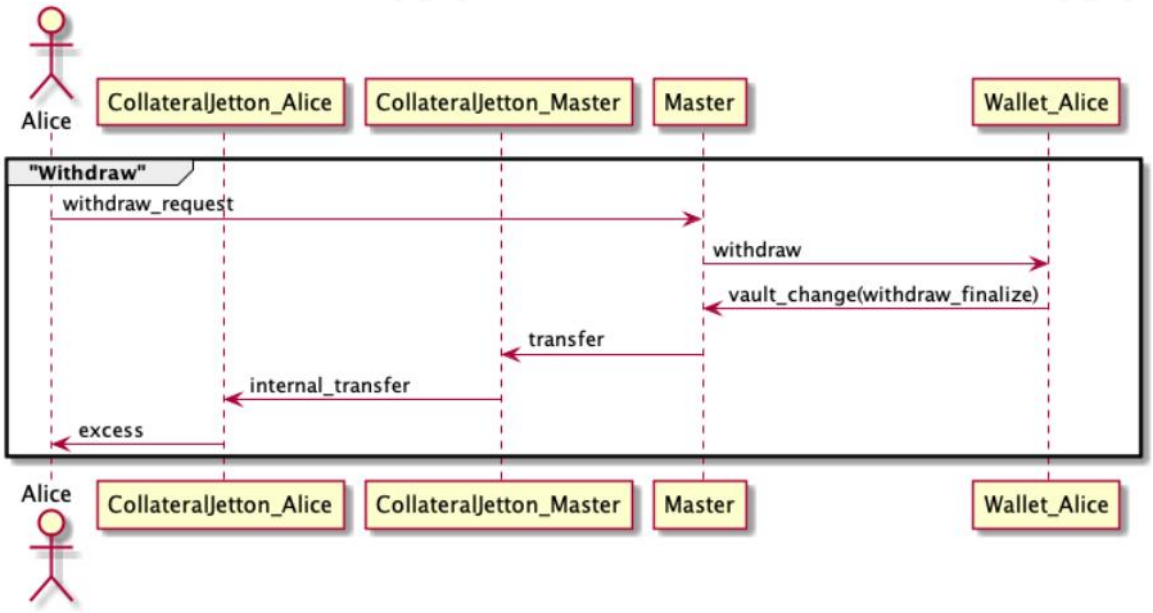



Figure 2 Screenshot of the Withdraw process

- Repay function explained:

First, Alice invokes the Aqua-Master contract to initiate a repay request. Once the Aqua-Master contract processes Alice's repay request, it sends a message to Alice's Aqua-wallet. Upon receiving the message, Alice's Aqua-wallet settles Alice's borrowing fees and then deducts the balance from Alice's Aqua-wallet to repay her debts. Finally, the Aqua-wallet sends a repayment completion notification to the Aqua-Master contract, which is used to update the key data such as total_supply, asset_total_supply, total_collateral_locked, and fee_total.

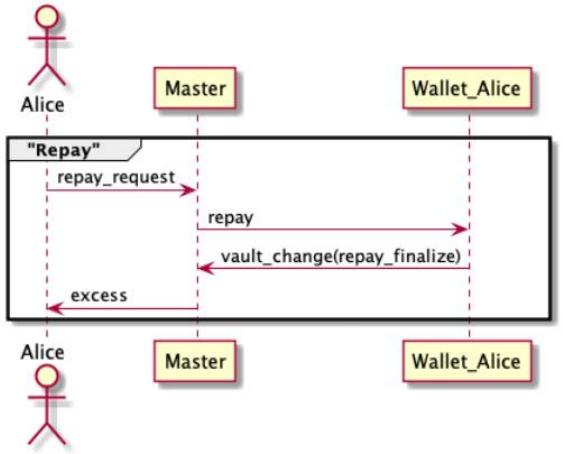



Figure 3 Repay Process Screenshot

➢ Redeem function explained:

Alice initiates a redeem request through the Aqua-Master contract, specifying the assets she wishes to redeem. Upon processing the request, the Aqua-Master sends a message to Alice's Aqua-wallet. The Aqua-wallet, in response, deducts the corresponding amount of aqua (debt) and notifies the Aqua-Master of the successful redemption confirmation. The Aqua-Master then retrieves data from the vaults dictionary, iterates through and locks relevant wallets, deducting the necessary redeem_amount and collateral_amount until Alice's redemption requirements are met. Finally, the redeemed assets are transferred from the master's Jetton wallet to Alice's Jetton wallet.

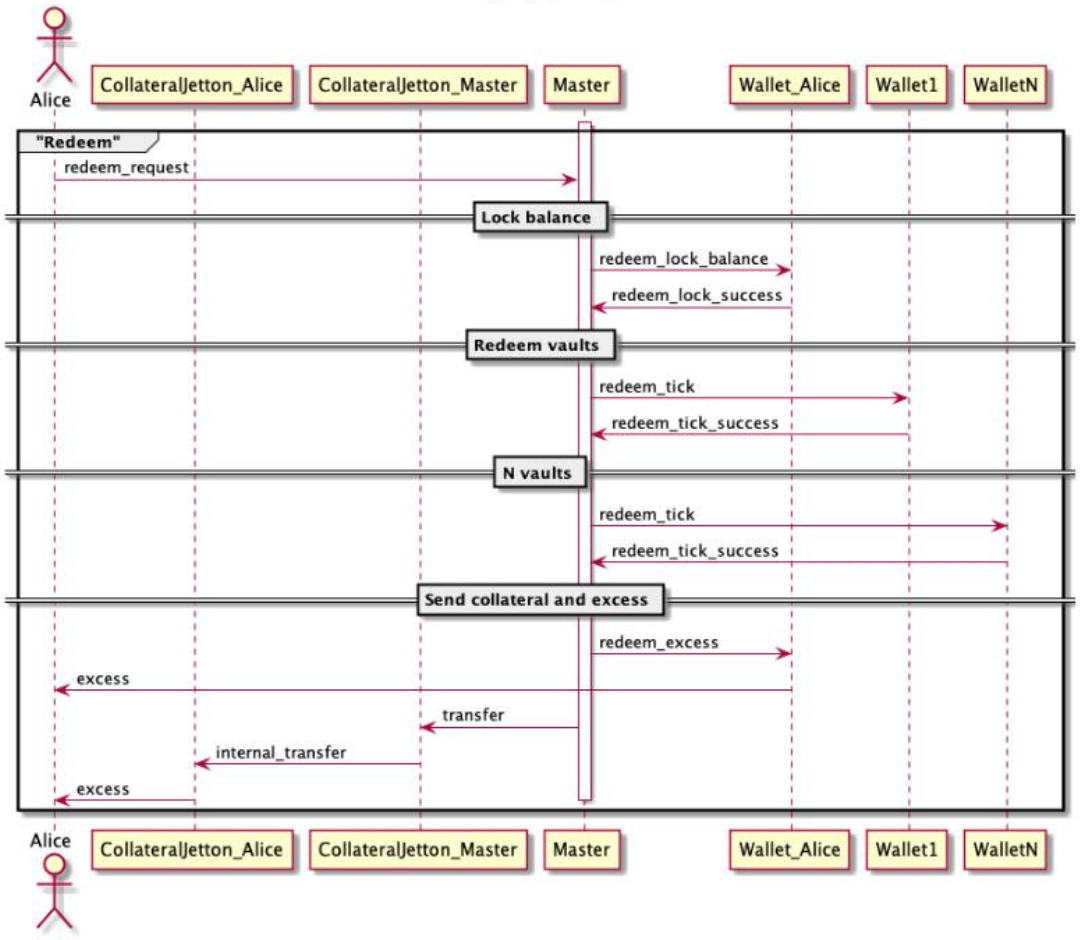



Figure 4 Redeem Process Screenshot

➢ Liquidate function explained:

Alice initiates a liquidate_request to the master actor through a signature from the keeper. This signature contains the address of the person being liquidated and the type of asset to be liquidated. Upon parsing the corresponding data, the master forwards a liquidate_init message to the caller's wallet to check for sufficient aquaUSD balance for liquidation and reduce the corresponding `payment_amount`. The message is then forwarded to the liquidatee's wallet for liquidation.

Upon receiving this message, the liquidatee's wallet updates the latest borrowing interest based on the payload and calculates whether the specified asset is in a bad_health state according to the collateral_price. If the conditions for full_liquidation are met (the project has bad debt and the liquidatee's borrowing ratio is below the full_liquidation_ratio), the liquidatee's collateral will be divided into two parts: one part is the liquidator's gain (the total collateral amount of the specified asset minus the keeper's reward), and the other part is the keeper's reward.

For partial liquidation of debt, there are two scenarios. In the first scenario, the bad debt increases after liquidation (the liquidator receives the liquidation reward, reducing the collateral amount), and the keeper receives no reward. In the second scenario, the debt becomes healthier, and both the liquidator and the keeper receive rewards. After completion, an op::liquidate_finalize message is forwarded to the master to update the global ledger state.

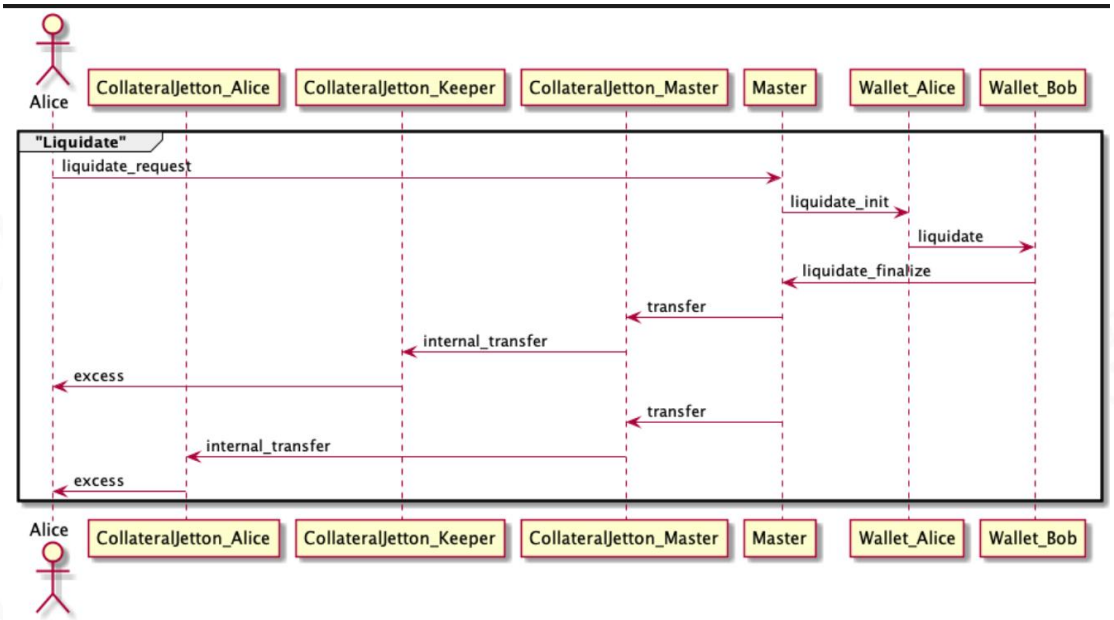



Figure 5 Liquidate Process Screenshot

# 1 Overview

## 1.1 Project Overview

| | |
|---|---|
| **Project Name** | Aqua Protocol |
| **Project Language** | Func |
| **Platform** | The Open Network |
| **Code base** | https://github.com/aquaprotocolxyz/contracts/tree/master |
| **Commit** | 0eb63eb8a278c400e42dae7af4a87f69f7870f74(initial)<br>a196732750e41c7ef98c7a8a24ad125e232890c1<br>8164241c264a36ea3a897ae1d40452320428c77c<br>d4743fbf86a01c55cc3336791ca7029c0813f562(final) |

## 1.2 Audit Overview

Audit work duration: July1, 2024 – July 22, 2024

Audit team: Beosin Security Team

## 1.3 Audit Method

The audit methods are as follows:

1. Formal Verification

Formal verification is a technique that uses property-based approaches for testing and verification. Property specifications define a set of rules using Beosin's library of security expert rules. These rules call into the contracts under analysis and make various assertions about their behavior. The rules of the specification play a crucial role in the analysis. If the rule is violated, a concrete test case is provided to demonstrate the violation.

2. Manual Review

Using manual auditing methods, the code is read line by line to identify potential security issues. This ensures that the contract's execution logic aligns with the client's specifications and intentions, thereby safeguarding the accuracy of the contract's business logic.

The manual audit is divided into three groups to cover the entire auditing process:

The Basic Testing Group is primarily responsible for interpreting the project's code and conducting comprehensive functional testing.

The Simulated Attack Group is responsible for analyzing the audited project based on the collected historical audit vulnerability database and security incident attack models. They identify potential attack vectors and collaborate with the Basic Testing Group to conduct simulated attack tests.

The Expert Analysis Group is responsible for analyzing the overall project design, interactions with third parties, and security risks in the on-chain operational environment. They also conduct a review of the entire audit findings.

3. Static Analysis

Static analysis is a method of examining code during compilation or static analysis to detect issues. Beosin-VaaS can detect more than 100 common smart contract vulnerabilities through static analysis, such as reentrancy and block parameter dependency. It allows early and efficient discovery of problems to improve code quality and security.

# 2 Findings

| Index | Risk description | Severity level | Status |
| --- | --- | --- | --- |
| Aqua Protocol-01 | The on_bounce is poorly designed | High | Fixed |
| Aqua Protocol-02 | Incorrect asset_address import location | Medium | Fixed |
| Aqua Protocol-03 | The mint function lacks permission checking | Low | Fixed |
| Aqua Protocol-04 | Error message | Info | Fixed |
| Aqua Protocol-05 | Redundant code | Info | Fixed |

# Finding Details:

## [Aqua Protocol-01] The on_bounce is poorly designed

| | |
|---|---|
| **Severity Level** | High |
| **Type** | Business Security |
| **Lines** | Master/handlers.func#L15-28 |
| **Description** | When a user utilizes the `redeem` function, the redemption data is stored in the master contract's `ctx::redeem_data`. If the message call for the redemption operation fails, the system redirects the error to the `on_bounce` function for handling. Within the `on_bounce` function, if the detected operation type is `op::redeem_lock_balance`, it clears the `ctx::redeem_data`, which is a reasonable approach. However, the entire redemption process also involves the processing of `redeem_tick` messages. If an exception or error occurs during the handling of `redeem_tick` messages, it does not trigger the `on_bounce` function, potentially leading to the persistence of data in `ctx::redeem_data` within the contract context without being promptly cleared.<br>If the user subsequently attempts another redemption operation, the presence of residual data from the previous redemption in `ctx::redeem_data` will cause an error when invoking redeem_request, as `redeem_request` requires `ctx::redeem_data` to be empty for execution. |

```
() handle::on_bounce(slice sender_address, slice in_msg_body) impure
inline {
  in_msg_body~skip_bounce_flag();
  int op = in_msg_body~load_op();
  ctx::query_id = in_msg_body~load_query_id();
  ;; @todo op::redeem_tick when collateral_amount >
  throw_unless(error::unknown_action_bounced, (op ==
op::redeem_lock_balance));
  if (op == op::redeem_lock_balance) {
    var (redeemer_address, _, _, _, _, vaults_dict) =
unpack_redeem_data(ctx::redeem_data);
    ctx::redeem_data = begin_cell().end_cell();
    send_excesses(redeemer_address);
    release_locks(vaults_dict);
```

```
  }
}
```

**Recommendation** It is recommended to extend the exception handling in the `on_bounce` function to cover all operations related to the redemption process (op).

**Status** **Fixed.**

```
() handle::on_bounce(slice sender_address, slice in_msg_body) impure
inline {
  in_msg_body~skip_bounce_flag();
  int op = in_msg_body~load_op();
  ctx::query_id = in_msg_body~load_query_id();
  throw_unless(error::unknown_action_bounced, (op ==
op::redeem_lock_balance) | (op == op::redeem_tick));
  if (op == op::redeem_lock_balance) {
    var (redeemer_address, _, _, _, _, _) =
unpack_redeem_data(ctx::redeem_data);
    ctx::redeem_data = begin_cell().end_cell();
    send_excesses(redeemer_address);
  }
  if (op == op::redeem_tick) {
    (slice redeemer_address, slice asset_master, int redeem_amount, int
collateral_redeemed, _, _) = unpack_redeem_data(ctx::redeem_data);
    ctx::redeem_data = begin_cell().end_cell();
    unlock(sender_address);
    if (redeem_amount > 0) {
      slice wallet_address = calc_user_wallet(redeemer_address,
my_address(), ctx::wallet_code);
      send_redeem_excess(wallet_address, redeem_amount);
    }
    if (collateral_redeemed) {
      load_asset_by_master(asset_master);
      send_collateral(ctx::wallet, redeemer_address,
collateral_redeemed);
    }
  }
}
```

## [Aqua Protocol-02] Incorrect asset_address import location

| | |
|---|---|
| **Severity Level** | **Medium** |
| **Type** | Business Security |
| **Lines** | Master/handlers.func#L247-260 |
| **Description** | In the TON (The Open Network) environment, due to the asynchronous nature of messaging, when a user performs a `redeem_tick` operation, new asset_address data may be imported if another user performs an operation such as mint beforehand. For example, if the `send_collateral` function in the `redeem_tick` function does not make a call to load_asset_by_master(asset_address) to update the `ctx::wallet address`, the redeemed assets may not match the actual assets. |
| **Recommendation** | It is recommended to move the line `(cell asset_settings_ref, _) = load_asset_by_master(asset_address);` to the top of the `redeem_tick` function. |
| **Status** | **Fixed.** |

```
() handle::redeem_tick() impure inline_ref {
  (slice redeemer_address, slice asset_address, int redeem_amount, int
collateral_redeemed, int collateral_price, cell vaults_dict) =
unpack_redeem_data(ctx::redeem_data);
  (int key, cell v, int f) = vaults_dict.udict_get_min_ref?(16);
  ;; order fulfilled or not vaults left
  if ((f == 0) | (redeem_amount == 0)) {
    send_collateral(ctx::wallet, redeemer_address,
collateral_redeemed);
    ctx::redeem_data = begin_cell().end_cell();
    return ();
  }
  (cell asset_settings_ref, _) = load_asset_by_master(asset_address);
  cell borrowing_rate_ref = pack_borrowing_rate_ref();
```

## [Aqua Protocol-03] The mint function lacks permission checking

| | |
|---|---|
| **Severity Level** | Low |
| **Type** | Business Security |
| **Lines** | jetton-master.func#L70-88 |
| **Description** | In the `mint` function of the jetton-master contract, the absence of permission checks allows anyone to mint tokens through the function, posing a risk of unauthorized token creation. |

```
    if (op == op::mint) {
        slice to_address = in_msg_body~load_msg_addr();
        int jetton_amount = in_msg_body~load_coins();
        int forward_ton_amount = in_msg_body~load_coins();
        int total_ton_amount = in_msg_body~load_coins();
        throw_unless(error::discovery_fee_not_matched,
total_ton_amount > forward_ton_amount);
        cell mint_request = begin_cell()
                .store_op(op::internal_transfer)
                .store_query_id(query_id)
                .store_coins(jetton_amount) ;; max 124 bit
                .store_uint(0, 2) ;; from_address, addr_none$00
                .store_slice(my_address())
                .store_coins(forward_ton_amount)
                .store_uint(0, 1) ;; no forward_payload, 1 bit
                .end_cell();
        mint_tokens(to_address, jetton_wallet_code, total_ton_amount,
mint_request);
        save_data(total_supply + jetton_amount, admin_address, content,
jetton_wallet_code);
        return ();
    }
```

| | |
|---|---|
| **Recommendation** | It is recommended to incorporate a permission check within the `mint` function to ensure that only authorized users can execute the minting operation. |
| **Status** | **Fixed.** Description of Project Parties: The contracts located in collateral-jetton/* are solely intended for testing purposes, aimed at assisting users in minting jettons within a testnet environment.It is recommended to modify the corresponding minting logic before the project goes online. |

## [Aqua Protocol-04] Error message

| | |
|---|---|
| **Severity Level** | Info |
| **Type** | Coding Conventions |
| **Lines** | Wallet/handlers.func#L247,L37,55 |
| **Description** | In aqua-wallet, wallet_balance should reflect the number of aqua, not the number of jetton, to maintain consistency with the output of the other functions, otherwise it will cause confusion in the error message.<br>`() handle::redeem_lock_balance(slice sender_address, int redeem_amount) impure inline {`<br>`  throw_unless(error::unauthorized_redeem_lock_balance, equal_slice_bits(ctx::master_address, sender_address));`<br>`  throw_unless(error::not_enough_jettons, ctx::wallet_balance >= redeem_amount);`<br>`  ctx::wallet_balance -= redeem_amount;`<br>`  send_redeem_lock_success(ctx::owner_address);`<br>`  return ();`<br>`}` |
| **Recommendation** | It is recommended to modify 'not_enough_jettons' to 'not_enough_aqua_in_wallet'. |
| **Status** | **Fixed.**<br>`() handle::redeem_lock_balance(slice sender_address, int redeem_amount) impure inline {`<br>`  throw_unless(error::unauthorized_redeem_lock_balance, equal_slice_bits(ctx::master_address, sender_address));`<br>`  throw_unless(error::not_enough_aqua_in_wallet, ctx::wallet_balance >= redeem_amount);`<br>`  ctx::wallet_balance -= redeem_amount;`<br>`  send_redeem_lock_success(ctx::owner_address);`<br>`  return ();`<br>`}` |

## [Aqua Protocol-05] Redundant code

| | |
|---|---|
| **Severity Level** | Info |
| **Type** | Coding Conventions |
| **Lines** | handlers.func#L188,185 |
| **Description** | The value of wallet_address is double-counted in the handle::withdraw_request function, which is redundant code.<br>`slice wallet_address = calc_user_wallet(sender_address, my_address(), ctx::wallet_code);`<br>`check_lock(wallet_address);`<br>`slice wallet_address = calc_user_wallet(sender_address, my_address(), ctx::wallet_code);` |
| **Recommendation** | It is recommended that redundant code be removed. |
| **Status** | **Fixed.** The project has removed redundant code. |

# 3 Appendix

## 3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

### 3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1(Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

| Likelihood \ Impact | Severe | High | Medium | Low |
|---|---|---|---|---|
| Probable | Critical | High | Medium | Low |
| Possible | High | Medium | Medium | Low |
| Unlikely | Medium | Medium | Low | Info |
| Rare | Low | Low | Info | Info |

### 3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

### 3.1.3 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

### 3.1.4 Fix Results Status

| Status | Description |
| --- | --- |
| **Fixed** | The project party fully fixes a vulnerability. |
| **Partially Fixed** | The project party did not fully fix the issue, but only mitigated the issue. |
| **Acknowledged** | The project party confirms and chooses to ignore the issue. |

## 3.2 Audit Categories

| No. | Categories | Subitems |
|---|---|---|
| 1 | Coding Conventions | Redundant Code |
| | | Deprecated Items |
| | | Gas Consumption* |
| | | Event Trigger |
| | | Throw Usage |
| 2 | General Vulnerability | Message Forgery* |
| | | Restore on Failure* |
| | | Integer Overflow/Underflow |
| | | Pseudo-random Number Generator (PRNG) |
| | | Transaction-Ordering Dependence |
| | | DoS (Denial of Service)* |
| | | Function Call Permissions |
| | | Message Flow Error* |
| | | Returned Value Security |
| | | Data Structure Error* |
| | | Replay Attack |
| | | Overriding Variables |
| | | Third-party Protocol Interface Consistency |
| 3 | Business Security | Business Logics |
| | | Business Implementations |
| | | Missing Calibration |
| | | Manipulable Token Price |
| | | Centralized Asset Control |
| | | Asset Tradability |
| | | Arbitrage Attack |

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in FunC language should strictly check for gas consumption and trigger events on critical changes.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

**TON Features**

*Gas Consumption:

Contracts should be strictly checked for gas, so that execution does not fail due to insufficient gas in a TON to execute all the code, and that code already executed is not rolled back.

*Message Forgery:

In FunC there are parent and child contracts that need to verify messages between the parent and child, and if a forged message is accepted there may be risks such as arbitrary coin minting by an attacker.

*Restore on Failure:

When message processing fails, the contract should throw an exception and senders with a fallback flag set should fall back, otherwise a partial execution of the transaction may occur, resulting in a loss of assets.

*DoS (Denial of Service):

As TON supports asynchronous execution, the nature of the func programming language can introduce contention conditions and logic errors due to asynchronous and threaded execution, which can lead to denial of service vulnerabilities.

*Message Flow Error:

In FunC, there are message calls between contracts and the message flow should be checked rigorously for design compliance, otherwise unexpected errors and losses can be introduced.

*Data Structure Error:

When calling the set_data function, you need to pay attention to the order of the arguments, otherwise it may lead to confusing data stored in the contract and seriously affect the business logic.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

## 3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

## 3.4 About Beosin

Beosin is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions. Beosin has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, Beosin has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security


**Official Website**
https://www.beosin.com

**Telegram**
https://t.me/beosin

**Twitter**
https://twitter.com/Beosin_com

**Email**
service@beosin.com