UTonic
Audit Report

Sat Oct 12 2024

contact@bitslab.xyz https://twitter.com/tonbit_

TonBit

UTonic Audit Report

1 Executive Summary

1.1 Project Information

Description	The First TON Restaking Protocol with Triple Yields
Type	Staking
Auditors	TonBit
Timeline	Fri Sep 27 2024 - Sat Oct 12 2024
Languages	FunC
Platform	Ton
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/UTONICFinance/utonic-contracts
Commits	f845e41674f41f901a8e7ac84d2ef9fd76722324 836726317bb0e8f2a0119d00f69fb71a34bf3a85 4ecd43bf53cf23b8cc1dede66d5714955f58b524 75dd56bcd899d3e2801e0cc3fe0cc1e3dd3d312a

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
STD	contracts/imports/stdlib.fc	2f104cd568a4cebb1c4112ecf8979800f0672575
EVE	contracts/proxy/events.func	aca46b574dbcf49b45a02677209be0cc0a60d83e
PAR	contracts/proxy/proxy_lst_ton/params.func	e015dbea7ac114bc7daa32fab418bf020e81fa15
ERR	contracts/proxy/proxy_lst_ton/errors.func	3c36549eadfcbd6632d2922de5a1cfca6f78fef7
CPPW2PF	contracts/proxy/proxy_whale2/params.func	afcc174d70ed23e0b63813a0ab25ecdac1b07d46
CPPW2OF	contracts/proxy/proxy_whale2/op.func	16a81ad17c0a95422144285259a6b0b1c95630d6
CPPTWOF	contracts/proxy/proxy_ton/withdraw/op.func	bc71f6faa71bd4fc757ae247b51f1eca7734eb26
CPPTWEF	contracts/proxy/proxy_ton/withdraw/errors.func	955acf8e3e10639783f4eb7de49bd3ed51dfc348
CPPTPF	contracts/proxy/proxy_ton/params.func	b288253d3afcacfa739f0165335514611ac681ad
CWPF	contracts/wallet/params.func	0f85983d103679417e325addd7853de766224243
CWSF	contracts/wallet/storage.func	41f440658bffb874c59576f28848123d6dc45b98

CWEF	contracts/wallet/errors.func	09ba6584fdd2197dc57cb696c52b2b1c977f4e2a
CLUF	contracts/libs/utils.func	37c2117f7f0fb9c5e7ee229f6f2e73dc9d333614
CMEF	contracts/minter/events.func	185552aab258d6b6486b2d6d4de50c9a5a2376e7
TYP	contracts/minter/types.func	1116320bcabe7a4f3ce19459e89b637a3152b376
CMEF	contracts/minter/errors.func	e2f807de2081e62c44df7244d9d8a7ee5ac175e9
CWF	contracts/wallet.func	c1cbebd12fa6934c6da40c3dce4008452d1e0b07
CCPF	contracts/common/params.func	61a6f72c31a3038281dd07dd42914921466541cf
CSOF	contracts/standard/op.func	54552571425d29862990920ab668a8159b6639ec
STO	contracts/proxy/proxy_lst_ton/storage.func	ef87244b55d893c6ade698bde0ed940ae99d5523
OP	contracts/proxy/proxy_lst_ton/op.func	44b3f1ce4910a6cdaa8e833f88b1e006e5b6f235
STO1	contracts/proxy/proxy_whale2/storage.func	9e25b08626580c80734fa175ea5ac9c9cdb98055
PW2	contracts/proxy/proxy_whale2/proxy_whale2.func	aa82f2d3d6a9ce167386c99fc997792ed0f01326
UTI	contracts/proxy/proxy_ton/withdraw/utils.func	9485632a51f3b64f5c5653c5a82dd6e7967cc816

STO2	contracts/proxy/proxy_ton/withdraw/storage.func	388131f83024d8a2ffb2b1d0532009dfdd0fbf03
PTO	contracts/proxy/proxy_ton/proxy_ton.func	17d5c3896816a1d66954fce48576f5f28e93079c
STO3	contracts/proxy/proxy_ton/storage.func	6598cee3162365f333f20d1f9bdd1d0014b1e672
OP3	contracts/proxy/proxy_ton/op.func	2c99197e064821d8b6f04af4334555d69fe52fbf
PAR7	contracts/minter/params.func	d0bae9dc558cd153cd0fc7af1960769ebe22057b
STO7	contracts/minter/storage.func	5bbded139224ebe0094f601e25e811e49247ee04
OP6	contracts/minter/op.func	525e74ff4ff131f461574267fce51687281c916c
UTI3	contracts/common/utils.func	366bd312b95b0b726b391f792e2262c15536fd93
OP7	contracts/common/op.func	7b0f61e033cbb87817b55fdbe556f6fdda40e0b6
ERR5	contracts/common/errors.func	17d06cdc21cc479be250a2680a39864cf1195004
MIN1	contracts/minter.func	25cc4dd2b8214ba8edcf7780be5723fa576ffbbc
PAR2	contracts/proxy/proxy_ton/withdraw/params.func	a9d918d08194b2faa1225e1d7aec7d01cda7ec21
WIT	contracts/proxy/proxy_ton/withdraw/withdraw.func	50997be4ad0ad4270df9ed72ab5c7bccb6ddead0

PLT	contracts/proxy/proxy_lst_ton/proxy_lst_ton.func	356469ea4c31bfbee69a06a4d78731bd92c6af2e

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	6	6	0
Informational	0	0	0
Minor	1	1	0
Medium	2	2	0
Major	3	3	0
Critical	0	0	0

1.4 TonBit Audit Breakdown

TonBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by UTonic to identify any potential issues and vulnerabilities in the source code of the UTonic smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 6 issues of varying severity, listed below.

ID	Title	Severity	Status
MIN-1	Front-running the price Update Allows Users to Consistently Make a Profit	Major	Fixed
MIN-2	Single-step Ownership Transfer Can be Dangerous	Medium	Fixed
MIN-3	Restrict proxy_id to Proxies Capable of Handling Burn Messages	Minor	Fixed
PLT-1	Refund User's LST Ton on Operation Failure	Major	Fixed
WIT-1	Fee Calculation Error	Medium	Fixed
MIN1-1	Centralization Risk	Major	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the UTonic Smart Contract :

User

- User can transfer uTON via the message `op == JETTON::OP::TRANSFER` .
- User can burn uTON to exchange for TON via the message `op == JETTON::OP::BUR` .
- User can transfer lst_ton to mint uTON via the message `op == JETTON::OP::TRANSFER_NOTIFICATION` .
- User can withdraw TON via the message `op == WITHDRAW::OP::WITHDRAW` .
- User can mint uTON via the message `op == COMMON::OP::STAKE` .

Admin

- Admin can change the admin via the message `op == PROXYLST::UPDATE_ADMIN` .
- Admin can change `lst_ton_price` via the message `op == PROXYLST::UPDATE_PRICE` .
- Admin can change `lst_ton_wallet` via the message `op == PROXYLST::UPDATE_PROXYLST_WALLET` .
- Admin can change `lst_ton_receiver_address` via the message `op == PROXYLST::UPDATE_LST_TON_RECEIVER` .
- Admin can send lst_ton to `lst_ton_receiver_address` via the message `op == PROXYLST::SEND_LST_TON` .
- Admin can change the `capacity` size via the message `op == PROXYLST::UPDATE_CAPACITY` .
- Admin can change `ton_receiver_address` via the message `op == PROXY_TON::OP::UPDATE_RECEIVER` .
- Admin can send TON via the message `op == PROXY_TON::OP::SEND_TON` .
- Admin can change ton_receiver_address via the message `op == PROXY_WHALE2::OP::UPDATE_TON_RECEIVER` .
- Admin can change price via the message `op == MINTER::OP::UPDATE_PRICE` .
- Admin can change price via the message `op == MINTER::OP::UPDATE_PRICE_INC` .

- Admin can update the whitelist via the message `op == MINTER::OP::UPDATE_PROXY_WHITELIST`.
- Admin can upgrade via the message `op == MINTER::OP::UPDATE_CODE_AND_DATA`.

Whale

- Whale can change uton_receiver_address via the message `op == PROXY_WHALE2::OP::UPDATE_UTON_RECEIVER`.

4 Findings

MIN-1 Front-running the price Update Allows Users to Consistently Make a Profit

Severity: Major

Status: Fixed

Code Location:

contracts/minter.func#248-260

Descriptions:

When users stake and burn, the protocol exchanges based on the `price`. As long as the `price` has not changed, the longer a user stakes, the more `ton` they will receive upon burning.

```
;; calculate ton amount
  int timestamp = now();
  int today = get_day(timestamp);
  int price = get_price(last_price_day, last_price, price_inc, today);
  int ton_amount = get_ton_amount(uton_amount, price);
```

The admin has the ability to update the `price`.

```
  if (op == MINTER::OP::UPDATE_PRICE) {
    load_global_data();
    throw_unless(COMMON::ERR::UNAUTHORIZED, equal_slices(sender_address,
admin_address));

    int new_price = in_msg_body~load_uint(64);
    int new_price_inc = in_msg_body~load_uint(64);
    int today = get_current_day();
    last_price_day = today;
    last_price = new_price;
    price_inc = new_price_inc;
    save_global_data();
    return ();
  }
```

The issue arises when users can front-run the admin's price update, allowing them to burn before the price changes. This ensures that users consistently profit from their actions.

Suggestion:

It is recommended to perform a double check when the user withdraws.

Resolution:

The client have modified the withdrawal process. When users withdraw, a callback mechanism will be used to obtain the latest price from the uTON-minter and take the minimum with the locked-in price at that time, in order to avoid this situation.

MIN-2 Single-step Ownership Transfer Can be Dangerous

Severity: Medium

Status: Fixed

Code Location:

contracts/minter.func#239-246

Descriptions:

Single-step ownership transfer means that if a wrong address was passed when transferring ownership or admin rights it can mean that role is lost forever. If the admin permissions are given to the wrong address within this function, it will cause irreparable damage to the contract. If `op == MINTER::OP::UPDATE_ADMIN`, the protocol directly changes `admin_address` to `new_admin_address`. This one-step transfer of admin rights poses the aforementioned risks.

```
  if (op == MINTER::OP::UPDATE_ADMIN) {
    load_global_data();
    throw_unless(COMMON::ERR::UNAUTHORIZED, equal_slices(sender_address,
admin_address));
    slice new_admin_address = in_msg_body~load_msg_addr();
    admin_address = new_admin_address;
    save_global_data();
    return ();
  }
```

Suggestion:

It is recommended to use a two-step ownership transfer pattern.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

MIN-3 Restrict proxy_id to Proxies Capable of Handling Burn Messages

Severity: Minor

Status: Fixed

Code Location:

contracts/minter.func#141-142

Descriptions:

In the uton minter contract, when `op == JETTON::OP::BURN_NOTIFICATION`, it checks whether the `proxy_id` is in the whitelist, as shown below:

```
(slice address_type, int has_address) = proxy_whitelist.udict_get?(32, proxy_id);
throw_unless(MINTER::ERR::INVALID_PROXY_ID, has_address);
```

There are currently three different proxies in the whitelist, but only `proxy_ton` can handle burn and withdraw messages. Therefore, proxy_id should be restricted to proxies that can process burn messages.

Suggestion:

It is recommended to restrict the `proxy_id` here to proxies that can handle burn messages.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

PLT-1 Refund User's LST Ton on Operation Failure

Severity: Major

Status: Fixed

Code Location:

contracts/proxy/proxy_lst_ton/proxy_lst_ton.func#85-144

Descriptions:

In `proxy_lst_Ton`, when the message `op == JETTON::OP::TRANSFER_NOTIFICATION`, the `proxy_stake` operation will be executed. However, if `proxy_stake` fails, for example, when the following condition check is not passed:

```
throw_unless(PROXYLST::ERR::CAPACITY_NOT_ENOUGH, capacity >= lst_ton_amount);
```

the user's lst Ton Jetton will still remain in Wallet_Proxy_lst_Ton. In such a case, consider refunding the user's lst Ton. Uncertain whether op == PROXYLST::SEND_LST_TON is intended to refund the user's lst Ton. Additionally, in the proxy_whale2 and proxy_ton contracts, if the minting process of uTon fails, there is still no operation to refund the user's Ton.

Suggestion:

It is recommended to refund the user's lst Ton in such cases.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

WIT-1 Fee Calculation Error

Severity: Medium

Status: Fixed

Code Location:

contracts/proxy/proxy_ton/withdraw/withdraw.func#101-109

Descriptions:

In the withdraw contract, when handling the message with `op == WITHDRAW::OP::WITHDRAW`, msg_value deducts fwd_fee as follows: `msg_value -= (storage_fee + WITHDRAW::WITHDRAW_FEE + fwd_fee);` however, the message sending mode is 0, which is incorrect. If fwd_fee is deducted, the message sending mode should be 1. Additionally, the msg_value check here is missing one fwd_fee in the calculation.

```
throw_unless(
    COMMON::ERR::INSUFFICIENT_VALUE,
    msg_value > MINTER::QUERY_FEE
        + fwd_fee
        + WITHDRAW::QUERY_ACK_FEE
        + fwd_fee
        + PROXY_TON::WITHDRAW_FEE
);
```

Suggestion:

It is recommended to follow the solution described in the explanation.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

MIN1-1 Centralization Risk

Severity: Major

Status: Fixed

Code Location:

contracts/test/jetton/minter.func#289-303

Descriptions:

The contract has a centralization risk issue where the administrator has the authority to arbitrarily upgrade the contract and modify prices. This can be observed in the following code snippet:

```
if (op == MINTER::OP::UPDATE_CODE_AND_DATA) {
    load_global_data();
    throw_unless(COMMON::ERR::UNAUTHORIZED, equal_slices(sender_address,
admin_address));
    int has_code = in_msg_body~load_uint(1);
    if (has_code) {
        cell code = in_msg_body~load_ref();
        set_code(code);
    }
    int has_data = in_msg_body~load_uint(1);
    if (has_data) {
        cell data = in_msg_body~load_ref();
        set_data(data);
    }
    return ();
}
```

This logic allows the administrator to update the contract code and data, which introduces a risk of centralized control and potential manipulation.

Suggestion:

It is recommended to use a multi-signature mechanism or similar methods to mitigate the centralization risk.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.