Pump Hub Audit Report

Mon Jul 29 2024

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

TonBit

Pump Hub Audit Report

1 Executive Summary

1.1 Project Information

Description	A Jetton factory protocol.
Type	DeFi
Auditors	TonBit
Timeline	Mon Jul 08 2024 - Mon Jul 29 2024
Languages	Tact
Platform	Ton
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/Pumphubbot/contracts
Commits	145d51bcd658f4fbe2ac1e870dfb37008dd40b84 65b3de7f82760983a4f4b3a9e7c48d86e1bd93c1 f024b9c05fff20f0358b78f67f8f27ad281081ea de55fc0c7f6c3452199ad20053347d1d09d61337

1.2 Files in Scope

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

ID	File	SHA-1 Hash
MJE	sources/contracts/PUMP/MessagesJetton.tact	1849db7686103ff3dba95291e2e1eb38af67c06a
JET	sources/contracts/PUMP/Jetton.tact	67e86c336fe77377d3b26605d6a550194cd0df4d
PUMP	sources/contracts/PUMP/PUMP.tact	e0cb7b4e656240cc30d4aa31f7e99e37cf9c3278
JOP	sources/contracts/PUMP/JettonOptimized.tact	52a22393aa0e0145816050d655e51ac39a0c6d66
FJE	sources/contracts/Factory/FactoryJetton.tact	f94319b4a8c2235c68636b32add832440de0d89e

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	5	5	0
Informational	0	0	0
Minor	1	1	0
Medium	1	1	0
Major	2	2	0
Critical	1	1	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 Pump Hub to identify any potential issues and vulnerabilities in the source code of the Pump Hub 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 5 issues of varying severity, listed below.

ID	Title	Severity	Status
FJE-1	The `total_supply` Value is Incorrect	Major	Fixed
FJE-2	Users Can Prematurely Claim	Major	Fixed
FJE-3	Replace + with Bitwise OR (|)	Medium	Fixed
FJE-4	The `bounce` Message was Not Handled	Minor	Fixed
PUM-1	Mint Jetton Lacks Access Control	Critical	Fixed

3 Participant Process

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

Owner

- The Owner can launch the Jetton through the `LaunchJetton` message.
- The Owner can launch the ton through the `LaunchTon` message.
- The Owner can withdraw funds through the `Withdraw` message.
- The Owner can set whether users can transfer Jetton through the `SetTransferable` message.

User

- Users can mint Jetton by spending ton through the `Mint` message.
- Users can claim Jetton through the `Claim` message.
- Users can transfer Jetton through the `TokenTransfer` message.
- Users can burn Jetton to redeem ton through the `TokenBurn` message.

4 Findings

FJE-1 The `total_supply` Value is Incorrect

Severity: Major

Status: Fixed

Code Location:

sources/contracts/Factory/FactoryJetton.tact#91-117

Descriptions:

When `LaunchJetton` was initiated, the total supply increased by the amount `ton("200000000")`. However, the actual amount transferred was `ADD_LIQUIDITY_JETTON: Int as coins = ton("2010")`, which does not match the increase in the total supply.

```
self.total_supply = self.total_supply + ton("200000000");
```

Suggestion:

It is recommended to adjust the `total_supply` to the correct value.

Resolution:

This issue has been fixed. The client has adjusted the `total_supply` to the correct value.

FJE-2 Users Can Prematurely Claim

Severity: Major

Status: Fixed

Code Location:

sources/contracts/Factory/FactoryJetton.tact#75-117

Descriptions:

`LaunchTon` and `LaunchJetton` both set `finish` to true, which means that regardless of whether `LaunchTon` or `LaunchJetton` is called first, the user can send a `Claim` message. This may not align with the protocol's design.

```
self.finish = true;
```

Suggestion:

It is recommended to set `finish to true` only after both `LaunchTon` and `LaunchJetton` have been executed.

Resolution:

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

FJE-3 Replace + with Bitwise OR (|)

Severity: Medium

Status: Fixed

Code Location:

sources/contracts/Factory/FactoryJetton.tact

Descriptions:

The current contract uses the `+` method in multiple instances to set the message sending mode, for example: in the `SetTransferable` function, the protocol sets the mode to `SendRemainingValue + SendIgnoreErrors`.

```
receive("SetTransferable") {
    let ctx: Context = context(); // Check sender
    require(ctx.sender == self.master, "Not the master");
    self.transferable = true;
    send(SendParameters{
        to: self.owner,
        value: 0,
        mode: SendRemainingValue + SendIgnoreErrors,
        bounce: false
    });
}
```

However, according to the official documentation, it is recommended to use the bitwise OR operator (|) instead.

```
Note, that while adding (+) base modes together with optional flags is possible, it is
discouraged due to the possibility of excess values. Use the bitwise OR (|) instead, as it's
designed to work with such flag and bit manipulations of the mode.
```

Suggestion:

It is recommended to use the bitwise OR (|) instead.

Resolution:

This issue has been fixed. The client has used the bitwise OR (|) instead.

FJE-4 The `bounce` Message was Not Handled

Severity: Minor

Status: Fixed

Code Location:

sources/contracts/Factory/FactoryJetton.tact#309-319

Descriptions:

When sending the `Claim` message, the `bounce` field was set to true, but the bounce message was not processed.

```
receive("Claim") {
    require(!self.transferable, "Has been claimed");
    // do not need to check if ctx.sender == owner
    send(SendParameters{
        to: self.master,
        value: 0,
        mode: SendRemainingValue + SendIgnoreErrors,
        bounce: true,
        body: Claim{ owner: self.owner}.toCell()
    });
}
```

Suggestion:

It is recommended to either handle the `bounce` message for this `Claim` message or set the `bounce` field to false.

Resolution:

This issue has been fixed. The client has set the `bounce` field to false.

PUM-1 Mint Jetton Lacks Access Control

Severity: Critical

Status: Fixed

Code Location:

sources/contracts/PUMP/PUMP.tact#27-31

Descriptions:

When processing `Mint` messages, there is no permission check, meaning anyone can mint any amount of Jettons.

```
receive(msg: Mint) {
    let ctx: Context = context();
    require(self.mintable, "Not mintable");
    self.mint(msg.to, msg.amount, msg.to);
}
```

Suggestion:

It is recommended to add access control to the `Mint` messages.

Resolution:

This issue has been fixed. The client has added access control to the `Mint` messages.

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