

PumpMeme Audit Report

Fri Nov 15 2024



contact@bitslab.xyz



https://twitter.com/tonbit_



PumpMeme Audit Report

1 Executive Summary

1.1 Project Information

Description	Ton Meme Launch Platform
Type	DeFi
Auditors	TonBit
Timeline	Thu Nov 14 2024 - Fri Nov 15 2024
Languages	Tact
Platform	Ton
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/pumpmeme/ton-contract
Commits	352439ef57e3429722cdd0f61cd47cd8dc7a17e11c7d698fe061fc873d5b62f9337bab18b6348522885e951499ef21c638faa3b376ffe6487c35d3

1.2 Files in Scope

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

ID	File	SHA-1 Hash
PME	contracts/pump_messages.tact	28ad51a7ab4791c3ef04ec9aed77e81185b223c6
PER	contracts/pump_errors.tact	ddb4631495d6efdf1401565a5ea209d9a3dbdbc4
JDW	contracts/jetton_default_wallet.tact	318f8183b017547b27d5473c50f72b539f13918f
PCO	contracts/pump_core.tact	216281a0032e53bf0062ddeaedc73c00b24b9821
JME	contracts/jetton_messages.tact	d2df9b4850e8af5a82d0ba39039e55f1a4743962
GAS	contracts/imports/gas.fc	1f9fe3194e4a1a5752c9be175cd4fdee2b2a9074
STD	contracts/imports/stdlib.fc	2f104cd568a4cebb1c4112ecf8979800f0672575
UTI	contracts/imports/utils.fc	127971aa7a1962e8d05a16f4ae8bb38098fdad71
TOK	contracts/token.tact	088dd3be1449e1e6496c38e5c2f282348542bcc3
JET	contracts/jetton.tact	b93f1fdbe459ff8fa2f854cba57718be2cf18e48

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	6	6	0
Informational	0	0	0
Minor	3	3	0
Medium	2	2	0
Major	1	1	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 [PumpMeme](#) to identify any potential issues and vulnerabilities in the source code of the [PumpMeme](#) 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
PCO-1	<code>BuyToken</code> Fee Issues	Major	Fixed
PCO-2	The Signature Does Not Include <code>content</code>	Medium	Fixed
PCO-3	Check If It Is Null Before Unpacking	Medium	Fixed
PCO-4	Code Optimization	Minor	Fixed
PCO-5	Parameter Settings Lack Range Checking	Minor	Fixed
PER-1	Same Error Code	Minor	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the [PumpMeme](#) Smart Contract :

Owner

- The owner can send a `WithdrawProtocolFee` msg to withdraw the protocol fee.
- The owner can send a `ChangeAdmin` msg to change the admin role
- The owner can send a `ChangePumpInfo` msg to change the pump info.
- The owner can send a `CreateDustDexDeposit` msg to create the dust dex pool.
- The owner can send a `ChangeFuncGasFee` msg to change the gas fee info.

User

- Users can send a `LaunchNewToken` msg to launch a new token.
- Users can send a `BuyToken` msg to buy tokens.

4 Findings

PCO-1 BuyToken Fee Issues

Severity: Major

Status: Fixed

Code Location:

contracts/pump_core.tact#123

Descriptions:

Users can buy tokens through the `BuyToken` msg, and the protocol will reduce the final real amount if either the number of ton exceeds the `targetTonAmount` , or the `tokenAmount` exceeds the `realNeedTonAmount` .

One of the issues is that if the user's actual payment is the `realNeedTonAmount` number of ton and the fee paid is calculated based on the `realNeedTonAmount` , but due to the existence of the `adjustSoldTokenAmount` function that will reduce the `tokenAmount` , it may result in the user finally getting a token value may be less than the actual paid value.

Suggestion:

It is recommended to ensure that this is in accordance with the protocol design.

PCO-2 The Signature Does Not Include content

Severity: Medium

Status: Fixed

Code Location:

contracts/pump_core.tact#502

Descriptions:

When creating a new pool, the user needs to pass in a legitimate signature, but the `checkSignature` function does not check `wrapContent` and `content`, resulting in the user being able to implement the `LaunchNewToken` operation multiple times with the same signature.

Suggestion:

It is recommended to make sure this is designed to comply with the protocol.

PCO-3 Check If It Is Null Before Unpacking

Severity: Medium

Status: Fixed

Code Location:

contracts/pump_core.tact#399

Descriptions:

Unpacking did not checks whether the `tokenInfo` is null,it may cause access to the field of the null value, which will cause an error.

```
let tokenInfoNull: TokenInfo? = self.tokenInfoMap.get(wjetton);
```

```
// Handle fund success but dex not created
if (tokenInfoNull!!!.fundSuccess && !tokenInfoNull!!!.dexCreate) {
    self.sendTokens(ctx.sender, msg.from,msg.amount);
    return;
}
```

Suggestion:

It is recommended to add additional safety checks.

PCO-4 Code Optimization

Severity: Minor

Status: Fixed

Code Location:

contracts/pump_core.tact#111

Descriptions:

When calculating the handling fee:

```
let buyFee = tonAmount * self.pumpInfo.protocolFeePercentage / 1000;
    let refundTonAmount: Int = 0;
    // Handle the last buy transaction, refund other ton
    if (tokenInfo.tonBalance + tonAmount - buyFee >= tokenInfo.targetTonAmount) { // the
amount of ton is enough this time
        let needTonAmount: Int = tokenInfo.targetTonAmount - tokenInfo.tonBalance; // real
need
        let realNeedTonAmount = needTonAmount * 1000 / (1000 -
self.pumpInfo.protocolFeePercentage); // plus protocol fee
        refundTonAmount = tonAmount - realNeedTonAmount; // left ton
        tonAmount = realNeedTonAmount;
    }

    buyFee = tonAmount * self.pumpInfo.protocolFeePercentage / 1000;
```

The fee is recalculated in the last line because the `buyFee` may have changed. If the `buyFee` has not changed, there is no need to recalculate the fee. In this case, the last line of code is redundant.

Suggestion:

It is recommended to optimize the code to:

```
let buyFee = tonAmount * self.pumpInfo.protocolFeePercentage / 1000;
    let refundTonAmount: Int = 0;
```

```
// Handle the last buy transaction,refund other ton
if (tokenInfo.tonBalance + tonAmount - buyFee >= tokenInfo.targetTonAmount) {// the
amount of ton is enough this time
    let needTonAmount: Int = tokenInfo.targetTonAmount - tokenInfo.tonBalance;// real
need
    let realNeedTonAmount = needTonAmount * 1000 / (1000 -
self.pumpInfo.protocolFeePercentage);// plus protocol fee
    refundTonAmount = tonAmount - realNeedTonAmount; //left ton
    tonAmount = realNeedTonAmount;
    buyFee = tonAmount * self.pumpInfo.protocolFeePercentage / 1000;
}
```

PCO-5 Parameter Settings Lack Range Checking

Severity: Minor

Status: Fixed

Code Location:

contracts/pump_core.tact#692;

contracts/pump_core.tact#695

Descriptions:

When setting `availableTokenPercentage` and `protocolFeePercentage`, the range of the parameters is not checked. If the parameters are accidentally set outside the range, this may cause unpredictable consequences for related calculations.

```
self.pumpInfo = PumpInfo {  
    commonTokenTotalSupply: msg.commonTokenTotalSupply,  
    targetTonAmount: msg.targetTonAmount,  
    availableTokenPercentage: msg.availableTokenPercentage,  
  
    protocolFeeReceiver: msg.protocolFeeReceiver,  
    protocolFeePercentage: msg.protocolFeePercentage,  
  
    createTokenFee: msg.createTokenFee,  
    pe: pe,  
    initTonAmountLimit: msg.initTonAmountLimit, // the first buy tx ton limit  
    signPublicKey: msg.signPublicKey,  
};
```

Suggestion:

It is recommended to add parameter range check.

PER-1 Same Error Code

Severity: Minor

Status: Fixed

Code Location:

contracts/pump_errors.tact#14

Descriptions:

`INVALID_INPUT_AMOUNT` and `INVALID_OUTPUT_AMOUNT` use the same error code and may not be distinguishable when the contract throws an exception.

Suggestion:

It is recommended to use the correct error code.

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.

