PUMPe-| |

Preliminary Audit Report

Thu Dec 26 2024

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

ScaleBit

PUMPe-|| Preliminary Audit Report

1 Executive Summary

1.1 Project Information

Description	This is a token lock smart contract.
Type	DeFi
Auditors	ScaleBit
Timeline	Wed Dec 25 2024 - Thu Dec 26 2024
Languages	Solidity
Platform	Ethereum
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/AnyAxis-Labs/evm-token-lock
Commits	426cdb4c9c7cacc2d2f2be57da0a2c73a10ddb49 751288c54dacce9cbbc1dcfd5fe24074037b9729

1.2 Files in Scope

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

ID	File	SHA-1 Hash
TLF	src/TokenLockFactory.sol	023be298fbd9753daeb078960ac05ca7e4b985e5
TLO	src/libs/TokenLock.sol	1d1f1a2410e09b1610ff81615608be879046e65e
TLO1	src/TokenLock.sol	8dcb1897b4ac5467c8ea19b6716c7c3d9a29a70c
IERC2	src/interfaces/IERC20.sol	c1c378726670b3bb450320b3351cea3504ec22c0

1.3 Issue Statistic

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

1.4 ScaleBit Audit Breakdown

ScaleBit 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
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** 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 Quang to identify any potential issues and vulnerabilities in the source code of the EvmTokenLock 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 3 issues of varying severity, listed below.

ID	Title	Severity	Status
TLF-1	No Access Control For `createTokenLock()`	Critical	Fixed
TLF-2	No Check for `transferFrom()` Return Value	Medium	Fixed
TLO-1	No Check for `transfer()` Return Value	Minor	Fixed

3 Participant Process

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

The User

- The user can create token lock through the function `createTokenLock()` .
- The user can claim token through the function `claim()` .

4 Findings

TLF-1 No Access Control For `createTokenLock()`

Severity: Critical

Status: Fixed

Code Location:

src/TokenLockFactory.sol#10

Descriptions:

The function `createTokenLock()` lacks access control and can be called by anyone. If a malicious user knows the address of `lock.depositor`, he can set his own address as `claimer`, lock the assets of `lock.depositor`, and obtain the assets that should belong to `lock.depositor`.

Suggestion:

1. Always verify the caller's identity
2. Implement appropriate access control mechanisms
3. Follow the "principle of least privilege"

Resolution:

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

TLF-2 No Check for `transferFrom()` Return Value

Severity: Medium

Status: Fixed

Code Location:

src/TokenLockFactory.sol#14

Descriptions:

While most tokens return true upon successful transfer and roll back transactions in case of transfer failure, there are still some tokens that return false when the transfer fails. If the return value is not checked, regardless of the success or failure of the transfer, it will be considered successful.

Suggestion:

It is recommended to add a return value check to the transfer operation , such as

```
require(IERC20(lock.token).transferFrom(
        lock.depositor,
        address(lockContract),
        lock.lockAmount
    ), "transfer failed");
```

Resolution:

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

TLO-1 No Check for `transfer()` Return Value

Severity: Minor

Status: Fixed

Code Location:

src/TokenLock.sol#41

Descriptions:

While most tokens return true upon successful transfer and roll back transactions in case of transfer failure, there are still some tokens that return false when the transfer fails. If the return value is not checked, regardless of the success or failure of the transfer, it will be considered successful.

Suggestion:

It is recommended to add a return value check to the `transfer` operation, such as

```
require(IERC20(lockInfo.token).transfer(
        lockInfo.claimer,
        claimable
    ), "transfer failed");
```

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.