BurveLOL
Audit Report

Fri Jun 21 2024

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

ScaleBit

BurveLOL Audit Report

1 Executive Summary

1.1 Project Information

Description	Burve Protocol is a consensus-driven token fair launch, swap, earn and AMM (Automated Market Maker) protocol serving multi-chains.
Type	AMM
Auditors	ScaleBit
Timeline	Wed Jun 19 2024 - Fri Jun 21 2024
Languages	Solidity
Platform	Ethereum
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/BurveProtocol/burve-contracts
Commits	2649d1bc74f345f9349bcc443a8a714165008e2f

1.2 Files in Scope

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

ID	File	SHA-1 Hash
BLOLB	src/preset/BurveLOLBsc.sol	877436550a53d57a65b29a9ed1f5756875faf3d0
BLOLB1	src/preset/BurveLOLBase.sol	209f653df467ba99cc63f94fdcabd2ff232bda3e
BLOL	src/preset/BurveLOL.sol	da619af3720e2aceecd856b11241081c96861ad5

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	3	1	2
Informational	0	0	0
Minor	0	0	0
Medium	2	0	2
Major	1	1	0
Critical	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 Burve to identify any potential issues and vulnerabilities in the source code of the BurveLOL 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
BLO-1	Division Before Multiplication Result in Precision Loss	Medium	Acknowledged
BLO1-1	Potential Token Left	Major	Fixed
BLO1-2	Use of `address(0xdead)` as Recipient	Medium	Acknowledged

3 Participant Process

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

N/A

4 Findings

BLO-1 Division Before Multiplication Result in Precision Loss

Severity: Medium

Status: Acknowledged

Code Location:

src/preset/BurveLOL.sol#46

Descriptions:

The line of code that calculates the `raisedAmount` is subject to potential precision loss due to the sequence of multiplication and division operations performed. Specifically, the calculation:

```
uint256 raisedAmount = _getBalance(raisingToken) - (((paidAmount * (10000)) / (10000 - _mintTax - pTax)) * (_mintTax + pTax)) / 10000;
```

performs the multiplication and division in a manner that can lead to rounding errors and precision loss.

Suggestion:

Avoid division before multiplication and only perform division at last.

Resolution:

The dev team acknowledges that they need to keep the tax before transferring.

BLO1-1 Potential Token Left

Severity: Major

Status: Fixed

Code Location:

src/preset/BurveLOLBase.sol#22;

src/preset/BurveLOLBsc.sol#22

Descriptions:

In the `_addLiquidity()` function, all tokens will be used to add liquidity on a third-party protocol. If the token ratio on the third-party protocol is inconsistent with expectations, it may result in a surplus of a particular asset, which will be locked in the current contract.

Suggestion:

It is recommended to take measures to avoid this issue.

Resolution:

The client modified the code to avoid transferring tokens to the third-party protocol before ido end and fixed this issue.

BLO1-2 Use of `address(0xdead)` as Recipient

Severity: Medium

Status: Acknowledged

Code Location:

src/preset/BurveLOLBase.sol#27,30;

src/preset/BurveLOLBsc.sol#27,30

Descriptions:

In the `_addLiquidity` function, `address(0xdead)` is used as the recipient address for liquidity tokens. While `address(0xdead)` is commonly used as a burn address, we are not sure if it's properly used in the `_addLiquidity` function.

Suggestion:

It is suggested to check if it's designed to be like this.

Resolution:

The dev team acknowledges that they intend to deposit the LP tokens into `dead` address.

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.