BlasterSwap

Security Audit Report

April 8, 2024

Contents

1 Introduction

1.1 About BlasterSwap

`BlasterSwap` is a native `Blast L2 DEX`, offering v2-style full-range liquidity. It eliminates trusted intermediaries and unnecessary forms of rent extraction, allowing for fast, efficient trading, besides leveraging `Blast L2` native features such as gas refund and liquidity auto rebase. It is forked from the `Uniswap V2` project with the support of gas refund and `WETH` and `USDB` balances rebasing on `Blast` network according to the instructions from Building on Blast documentation in the following links:

- https://docs.blast.io/building/guides/gas-fees
- https://docs.blast.io/building/guides/weth-yield

1.2 Source Code

The following source code was reviewed during the audit:

- https://github.com/blasterswap/core-v2/
- Commit ID: c5ca4f7

And this is the final version representing all fixes implemented for the issues identified in the audit:

- https://github.com/blasterswap/blasterswap-core-v2/
- Commit ID: bd67a30

2 | Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the `BlasterSwap` project. Throughout this audit, we identified a total of 3 issues of `Informational` severity level. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	-	-	-	-
High	-	-	-	-
Medium	-	-	-	-
Low	-	-	-	-
Informational	3	-	-	3
Undetermined	-	-	-	-

3 | Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Description
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and remediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

[I-1] Suggested Usage of Constant BLAST Address

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Coding Practices	NA	NA	Addressed

According to `Blast` documentation, smart contracts must interact with the `Blast` contract located at 0x4300000000000000000000000000000000000002 to change their Gas Mode. The contracts in `BlasterSwap` protocol interact with the `Blast` contract to change the gas mode. While reviewing the related multiple smart contracts logic, we notice that the current implementation directly uses address $0x430...002$ everywhere it needs to interact with the `Blast` contract. We think it need to be improved.

Specifically, as a good programming practice, we recommend defining the address $0x430...002$ as a constant variable (e.g., `BLAST`) so that this constant variable can be used wherever needed. It will also eliminate the possible inconsistency of multiple `Blast` contract reference.

BlasterswapV2Factory::constructor()

```
24 constructor(address _feeToSetter, address _governor) public {
25     feeToSetter = _feeToSetter;
26     governor = _governor;

28     BLAST = IBlast(0x4300000000000000000000000000000000000002);

30     BLAST.configureClaimableGas();
31     BLAST.configureGovernor(_governor);
32 }
```

Note the same suggestion is also applied to `BlasterswapV2Pair` and `BlasterswapV2Router02` contracts.

Remediation Properly define the address of `Blast` contract as a constant variable and replace the $0x430...002$ usage with it.

[I-2] Improved Validation for Governor in BlasterswapV2Factory

Target	Category	IMPACT	LIKELIHOOD	STATUS
BlasterswapV2Factory.sol	Coding Practices	NA	NA	Addressed

`BlasterSwap` protocol calls the `configureGovernor()` function of the `Blast` contract to set its `governor` address. After that, the `governor` can set the gas mode, claim gas fees and reconfigure the governor.

While examining its logic, we notice that the current implementation does not properly validate the input `governor` (line 26) argument. As a result, if the input `governor` is `address(0)`, the `governor` of the `BlasterSwap` protocol will not be changed at all. Based on this, we suggest to add proper validation for the input `governor` argument and ensure it is a valid address (i.e., `!address(0)`).

BlasterswapV2Factory::constructor()

```
24 constructor(address _feeToSetter, address _governor) public {
25     feeToSetter = _feeToSetter;
26     governor = _governor;

28     BLAST = IBlast(0x4300000000000000000000000000000000000002);

30     BLAST.configureClaimableGas();
31     BLAST.configureGovernor(_governor);
32 }
```

Remediation Add proper validation for the `governor` argument and ensure it's not `address(0)`.

[I-3] Improved Usage of IBlasterswapV2Pair in BlasterswapV2Library

Target	Category	IMPACT	LIKELIHOOD	STATUS
BlasterswapV2Library.sol	Business Logic	NA	NA	Addressed

As mentioned in the `Introduction` section, the `BlaserSwap` is forked from the `Uniswap V2` project with the support of gas refund and `WETH` and `USDB` balances rebasing on `Blast` network. It copies all the `Uniswap V2` smart contracts and the interface files into the `core-v2` repository with some adaption to `BlaserSwap` (i.e. BlasterSwap naming changes, etc). While reviewing the whole protocol, we noticed that there are two cases which reference to the `Uniswap V2` interface incorrectly.

To elaborate, we show below the `BlasterswapV2Library`. We notice that it imports `IUniswapV2Pair.sol` from `Uniswap V2` (line 3) which is incorrect. Note `IUniswapV2Pair.sol` has been renamed to `IBlasterswapV2Pair.sol` in the local interface directory and as a result, it should import `IBlasterswapV2Pair.sol` in the local repository here.

BlasterswapV2Library.sol

```
1   pragma solidity =0.6.6;

3   import "@uniswap/v2-core/contracts/interfaces/IUniswapV2Pair.sol";
4   import "./SafeMath.sol";

6   library BlasterswapV2Library {
7       using SafeMath for uint;

9       // returns sorted token addresses, used to handle return values from pairs
            sorted in this order
10      function sortTokens(
11          address tokenA,
12          address tokenB
13      ) internal pure returns (address token0, address token1) {
14          require(tokenA != tokenB, "BlasterswapV2Library: IDENTICAL_ADDRESSES")
                ;
15          (token0, token1) = tokenA < tokenB
16              ? (tokenA, tokenB)
17              : (tokenB, tokenA);
18          require(token0 != address(0), "BlasterswapV2Library: ZERO_ADDRESS");
19      }
```

Also the following reference to `IUniswapV2Pair` (line 49) shall be changed to `IBlasterswapV2Pair`.

BlasterswapV2Library::getReserves()

```
42    // fetches and sorts the reserves for a pair
43    function getReserves(
44        address factory,
45        address tokenA,
46        address tokenB
47    ) internal view returns (uint reserveA, uint reserveB) {
48        (address token0, ) = sortTokens(tokenA, tokenB);
49        (uint reserve0, uint reserve1, ) = IUniswapV2Pair(
50            pairFor(factory, tokenA, tokenB)
51        ).getReserves();
52        (reserveA, reserveB) = tokenA == token0
53            ? (reserve0, reserve1)
54            : (reserve1, reserve0);
55    }
```

Remediation Apply the necessary code changes and and reference `IBlasterswapV2Pair` instead of `IUniswapV2Pair` in the `BlasterswapV2Library` implementation.

4 Conclusion

BlasterSwap is forked from the Uniswap V2 project with the support of gas refund and WETH and USDB balances rebasing on Blast network. The current code base is well structured and neatly organized. Those identified issues were promptly confirmed and fixed.

Furthermore, we need to emphasize that smart contracts as a whole are still in exciting stage of development. To improve this report, we greatly appreciate any constructive feedback or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

5 Appendix

5.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

5.2 Disclaimer

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.

5.3 Contact

Name	AstraSec Team
Phone	+86 156 0639 2692
Email	contact@astrasec.ai
Twitter	https://twitter.com/AstraSecAI