BXHash V2 Audit Report

Version 1.0.0

Serial No. 20210716000062027

Presented by Fairyproof

July 16, 2021

FAIRYPROOF

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the BXHashV2 project, at the request of the BXHash team.

Audited Code's Github Repository:

https://github.com/BXHash/bxh_v2

Audited Code's Github Commit Number:

c301970493c81abe658ddb75468232c0ec9a74af

Audited Source Files' Onchain Address:

N/A

Audited Source Files:

The calculated SHA-256 values for the initial files are as follows:

```
BXHDaoPoolV2.sol :
0x8a415ec034a731556e311a0b3f756b042484fdda42de152ed9f9686469c4f031
Controller.sol   :
0xd8e8d8cdb278cbc1b4d255471efad6ea34727417afa6e42963af760f6d8b1532
DelegateERC20.sol:
0x36e247c440a4c4bf6870d1e5533d7f8f5f178563999f77a5c042aeb72cff74c7
GovController.sol:
0x67d7990731ec17b988d92a0c7c56f23475a21bb92611e1c68d6bf4da2b454266
PreWapper.sol    :
0x1dd1f46f06e372f23459c4c65db842ec6eb972647ca6c3c4b4ca8a1eea06fa15
Strategy.sol     :
0x976fd59cc6459bbe28254bf10f4bd35a16dd0e0d38be24497a91d5da62cc3ca0
StrategyDepth.sol:
0xb65d8f814d3c5f0edfe09bc68a89be83d77ea5c5633673f561d8befaa97c69a5
VaultPool.sol    :
0xf7cde0d4fbd2fdbdf9a51a6eaf05732497ac5b0580a146f83025c137ed01b5cb
XToken.sol       :
0x1d106b37d1821112eb541c861e83f159aca7fa912bd260d4eede6c6e7ec7b519
```

The source files audited include all the files with the extension "sol" as follows:

```
./
├── BXHDaoPoolV2.sol
├── Controller.sol
├── DelegateERC20.sol
├── GovController.sol
├── PreWapper.sol
├── Strategy.sol
├── StrategyDepth.sol
├── VaultPool.sol
└── XToken.sol

9 files
```

The goal of this audit is to review BXHashV2's solidity implementation for its aggregator application, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

— Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding system security. 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 sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from offchain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

— Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Fairyproof to make sure we understand the size, scope, and functionality of the project's source code.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Fairyproof describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run the test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the source code to improve maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

— Structure of the document

This report contains a list of issues and comments on all the above source files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

— Documentation

For this audit, we used the following sources of truth about how the BXHashV2 system should work:

https://github.com/BXHash/bxh_v2

These were considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the BXHash team or reported an issue.

— Comments from Auditor

No vulnerabilities with critical, high, medium or low-severity were found in the above source code.

02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying blockchain applications.

03. Introduction to BXHashV2

This is an aggregator application developed by the BXHash team for the BXHash Finance system.

When users stake a token in the application they will get its certification token. Users can stake this certification token in the application's DaoPool to earn profits.

Users' staked assets will be used by the BXHash team to make LPs and provide to third-party platforms to earn profits. The earned profits will be reinvested to third-party platforms earn further profits.

Attention:

- **Users need to hold certification tokens to take back their staked assets. There are some constraints applied to the trading of the certification tokens to prevent unexpected loss of assets.**
- **Users' staked assets will be used by the BXHash team to make LPs and earn profits. This may suffer impermanent loss which should be kept in mind and better be hedged.**

04. Major functions of audited code

The audited code implements the following functions:

1. `PreWapper.sol`: an interface for users to stake assets. When users use this interface to stake a token they will get a certification token for it. There are some contrains applied to the trading of the certification token to prevent unexpected loss of assets. Users need to hold this certification token to get back their staked assets.
2. `VaultPool.sol`: a vault for staked assets. Users can directly interact with this contract to stake or withdraw assets. But if users do it this way they will not get any profits or certification tokens.
3. `Controller.sol`: a controller contract. It is a bridge that connects vaults to investment strategies.
4. `GovController.sol`: this contract is used by the BXHash team to manage controller contracts, and collect profits or hedge losses.
5. `Strategy.sol`: this is an investment strategy contract that forms LPs and makes investment.
6. `StrategyDepth.sol`: this contract transfers assets to a Depth's aggregator, gets certification tokens from Depth and stakes the certification tokens to do mining.
7. `DelegateERC20.sol`: an abstract ERC-20 contract that utilizes Openzeppelin's library which has voting functions.
8. `XToken.sol`: a certification token contract. Note: this certification token is different from the one mentioned in point 1. There are constraints applied to the trading of this certification token as well to prevent unexpected loss. Users need to hold this certification token to withdraw their staked assets as well.
9. `BXHDAOPooV2.sol`: a contract that manages profits. After users stake assets they will get the certification tokens mentioned in point 8 and periodically get profits. Note: there is a locking period before users can get back their staked assets. And users need to hold certification tokens to get back their staked assets.

05. Key points in audit

During the audit Fairyproof worked closely with the BXHash team, reviewed possible vulnerabilities in access control, asset security etc and gave some suggestions to enhance the overall security.

06. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Re-entrancy Attack
- DDos Attack
- Integer Overflow
- Function Visibility
- Logic Vulnerability
- Uninitialized Storage Pointer
- Arithmetic Precision
- Tx.origin
- Shadow Variable
- Design Vulnerability
- Token Issurance
- Asset Security
- Access Control

07. Severity level reference

Every issue in this report was assigned a severity level from the following:

Critical severity issues need to be fixed as soon as possible.

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

08. Major areas that need attention

Based on the provided souce code the Fairyproof team focused on the possible issues and risks related to the following functions or areas.

- Asset Transfer and Security

We checked all asset transfer's design logic and checked whether or not there were potential issues or risks.

We didn't find issues or risks in this area at the time of writing.

- Access Control

We checked whether or not there was inappropriate access control to assets.

We didn't find issues or risks in this area at the time of writing.

- Code Implementation

We checked whether or not the code implementation had covered complete design logic.

We didn't find issues or risks in this area at the time of writing.

- Miscellaneous

We didn't find issues or risks in other functions or areas at the time of writing.

09. List of issues by severity

A. Critical

- N/A

B. High

- N/A

C. Medium

- N/A

D. Low

- N/A

10. List of issues by source file

- N/A

11. Issue descriptions

- N/A

12. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- N/A