Security Audit Report for XHedge

Date: Oct 15, 2021

Version: 1.1

Contact: contact@blocksecteam.com

Contents

Report Manifest

Item	Description
Client	Matrixport Technologies Ltd.
Target	XHedge

Version History

Version	Date	Description
1.0	Oct 15, 2021	First Release
1.1	Oct 16, 2021	Add fix commit hash

About BlockSec The BlockSec Team focuses on the security of the blockchain ecosystem, and collaborates with leading DeFi projects to secure their products. The team is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and released detailed analysis reports of high-impact security incidents. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

The target contract is A DeFi-Friendly PoS Scheme. It begins with a smart contract named XHedge which extends the function of AnyHedge (a famous Bitcoin Cash futures contract). Then it combines XHedge and the concept of coin-day to enable voting. The detailed description is in the following link: LeverDay: A DeFi-Friendly PoS Scheme.

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The files that are audited in this report include the following ones.

Repo Name	Github URL
xhedge	`https://github.com/smartbch/xhedge/blob/main/contracts/xhedge.sol`

The commit hash before the audit is `085c24d83c592e71aaa414b57d2ab19d0bb0d244`. The commit hash that fixes the issues found in this audit is `d309232857296f08298904db5d2821b870cc661e`.

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report do not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

- Reentrancy
- DoS
- Access control
- Data handling and data Flow
- Exception handling
- Untrusted external call and control flow
- Initialization consistency
- Events operation
- Error-prone randomness
- Improper use of the proxy system

1.3.2 DeFi Security

- Semantic consistency
- Functionality consistency
- Access control
- Business logic
- Token operation
- Emergency mechanism
- Oracle security
- Whitelist and blacklist
- Economic impact
- Batch transfer

1.3.3 NFT Security

- Duplicated item
- Verification of the token receiver
- Off-chain metadata security

1.3.4 Additional Recommendation

- Gas optimization

- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [1] and Common Weakness Enumeration [2]. Accordingly, the severity measured in this report are classified into four categories: **High**, **Medium**, **Low** and **Undetermined**.

[1] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[2] https://cwe.mitre.org/

Chapter 2 Findings

In total, we find one potential issue in the smart contract. We also have one recommendation, as follows:

- High Risk: 0
- Medium Risk: 0
- Low Risk: 1
- Recommendations: 1

ID	Severity	Description	Category
1	Low	*Lack of validation of parameters in* `createVault`	DeFi Security
2	-	*Pay attention to the implementation of the price oracle*	Recommendation

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Lack of validation of parameters in `createVault`

Status Confirmed and fixed.

Description The function `createVault` lacks the validation for the `initCollateralRate` and `matureTime`, i.e., initCollateralRate must be equal or greater than `minCollateralRate` and `matureTime` must be larger than current block time.

```
function createVault(
  uint64 initCollateralRate,
  uint64 minCollateralRate,
  uint64 closeoutPenalty,
  uint64 matureTime,
  uint validatorToVote,
  uint96 hedgeValue,
  address oracle) public payable {
  Vault memory vault;
  vault.initCollateralRate = initCollateralRate;
  vault.minCollateralRate = minCollateralRate;
  vault.closeoutPenalty = closeoutPenalty;
  vault.lastVoteTime = uint64(block.timestamp);
  vault.hedgeValue = hedgeValue;
  vault.matureTime = matureTime;
  vault.validatorToVote = validatorToVote;
  vault.oracle = oracle;
  uint price = PriceOracle(oracle).getPrice();
  uint amount = (10**18 + uint(initCollateralRate)) * uint(hedgeValue) / price;
  require(msg.value >= amount, "NOT_ENOUGH_PAID");
  require(amount >= GlobalMinimumAmount, "LOCKED_AMOUNT_TOO_SMALL");
  vault.amount = uint96(amount);
  if(msg.value > amount) { // return the extra coins
```

```
      safeTransfer(msg.sender, msg.value - amount);
    }
    uint idx = uint160(msg.sender) & 127;
    uint sn = nextSN[idx];
    nextSN[idx] = sn + 1;
    sn = (sn<<8)+(idx<<1);
    _safeMint(msg.sender, (sn<<1)+1); //the LeverNFT
    _safeMint(msg.sender, sn<<1); //the HedgeNFT
    saveVault(sn, vault);
  }
```

Impact The vault may not work as expected.

Suggestion Add validation for these parameters.

2.2 Additional Recommendation

2.2.1 Pay attention to the implementation of the price oracle

Status Acknowledged.

Description This smart contract highly leverages the price oracle to provide the current price of `BCH`. The price oracle is not included in this audit. We recommend that the project owner needs to ensure the robust, correctness and the security of the price oracle.

Impact NA

Suggestion Ensure the robust, correctness and the security of the price oracle.