BlockSec

Security Audit Report for ExchangeHub

Date: Jan 27, 2022

Version: 1.0

Contact: contact@blocksecteam.com

BlockSec
Audit Team

Digitally signed by
BlockSec Audit Team
Date: 2022.01.28 00:29:07
+08'00'

Contents

Report Manifest

Item	Description
Client	MatrixPort
Target	ExchangeHub

Version History

Version	Date	Description
1.0	Jan 27, 2022	First Release

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

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

The auditing process is iterative. Specifically, we will audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values of the repo [1] during the audit are shown in the following.

Contract Name	Stage	Commit SHA
ExchangeHub	Initial	e280770106f011c204cb91e532cde9c8b0555056
ExchangeHub	Final	06da941a0bd705aec56b665837b3003339bb6e35

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).

[1] https://github.com/fadingorders/fadingorders/commits/main/solidity/contracts/ExchangeHub.sol

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 [2] and Common Weakness Enumeration [3]. Accordingly, the severity measured in this report are classified into four categories: **High**, **Medium**, **Low** and **Undetermined**.

Furthermore, the status of a discovered issue will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The issue has been received by the client, but not confirmed yet.
- **Confirmed** The issue has been recognized by the client, but not fixed yet.
- **Fixed** The issue has been confirmed and fixed by the client.

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

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

Chapter 2 Findings

In total, we find two potential issues in the smart contract. We also have two recommendations, as follows:

- High Risk: 0
- Medium Risk: 0
- Low Risk: 2
- Recommendations: 2

ID	Severity	Description	Category	Status
1	Low	The newDueTime in addNewDueTime should be checked	Software Security	Fixed
2	Low	The return value of low level call should be checked	DeFi Security	Fixed
3	-	Lift the parameter sanitation to save gas	Recommendation	Fixed
4	-	Ensure that the exchange function will not be abused	Recommendation	-

The details are provided in the following sections.

2.1 Software Security

2.1.1 The newDueTime in addNewDueTime should be checked

Status Fixed

Description In function addNewDueTime , the newDueTiem should be checked to ensure that it's not zero. Otherwise, the link list of the due time will be corrupted.

```
function addNewDueTime(uint newDueTime) external {
 uint currTime = block.timestamp*MUL;
 clearOldDueTimesAndInsertNew(msg.sender, newDueTime, currTime);
}
```

Listing 2.1: ExchangeHub.sol

Impact The link list of the due time could be corrupted.

Suggestion Check whether newDueTiem is zero.

2.2 DeFi Security

2.2.1 Possible Loss with Incorrect Call Sequence

Status Fixed

Description The _exchange function uses the low-level call to transfer BCH to the makerAddr (line 193). However, it lacks the check for the return value of this call.

```
  function _exchange(uint256 coinsToMaker, uint256 coinsToTaker, uint256 takerAddr_dueTime80_v8,
      address makerAddr, bytes32 r, bytes32 s) private {
  if(makerAddr == address(0)) { //called by "exchange"
    makerAddr = getSigner(coinsToMaker, coinsToTaker, uint(uint160(0)),
    takerAddr_dueTime80_v8, r, s);
  } else { //called by "exchangeWithAgentSig"
    address agentAddr = getSigner(coinsToMaker, coinsToTaker, uint(uint160(makerAddr)),
    takerAddr_dueTime80_v8, r, s);
    require(makerToAgent[makerAddr] == agentAddr, "invalid agent");
  }
  uint dueTime = uint80(takerAddr_dueTime80_v8>>8);
  uint currTime = block.timestamp*MUL;
  require(currTime < dueTime, "too late");
  clearOldDueTimesAndInsertNew(makerAddr, dueTime, currTime);
  address takerAddr = address(bytes20(uint160(takerAddr_dueTime80_v8>>(80+8))));
  if(takerAddr == address(0)) { //if taker is not specified, anyone sending tx can be the taker
    takerAddr = msg.sender;
  }
  address coinTypeToMaker = address(bytes20(uint160(coinsToMaker>>96)));
  uint coinAmountToMaker = uint(uint96(coinsToMaker));
  address coinTypeToTaker = address(bytes20(uint160(coinsToTaker>>96)));
  uint coinAmountToTaker = uint(uint96(coinsToTaker));
  emit Exchange(makerAddr, coinsToMaker, coinsToTaker, takerAddr_dueTime80_v8>>8);\right)
  if(coinAmountToTaker != 0) {
    (bool success, bytes memory _notUsed) = coinTypeToTaker.call(
    abi.encodeWithSignature("transferFrom(address,address,uint256)",
    makerAddr, takerAddr, coinAmountToTaker));
    require(success, "transferFrom fail");
  }
  if(coinAmountToMaker != 0) {
    if(coinTypeToMaker == BCHAddress) {
      require(msg.value == coinAmountToMaker, "bch not enough");
      makerAddr.call{gas: 9000, value: coinAmountToMaker}("");
    } else {
      require(msg.value == 0, "no need for bch");
      IERC20(coinTypeToMaker).transferFrom(takerAddr, makerAddr, coinAmountToMaker);
    }
  }
}
```

Listing 2.2: ExchangeHub.sol

Impact The call could fail. However, the transaction will not revert.

Suggestion Check the return value and revert the transaction if the call fails.

2.3 Additional Recommendation

2.3.1 Lift the parameter sanitation to save gas

Status Fixed

Description In the `_exchange` function, the check for the `dueTime` and `currTime` (line 171-173) could be moved up before the invocation of `getSigner`. This can ensure that if the `dueTime` or `currTime` is invalid, the function will revert immediately without invoking expensive `getSigner` function. That's because the revert transaction will only return unused gas.

```
161 function _exchange(uint256 coinsToMaker, uint256 coinsToTaker, uint256 takerAddr_dueTime80_v8,
162   address makerAddr, bytes32 r, bytes32 s) private {
163   if(makerAddr == address(0)) { //called by "exchange"
164     makerAddr = getSigner(coinsToMaker, coinsToTaker, uint(uint160(0)),
165     takerAddr_dueTime80_v8, r, s);
166   } else { //called by "exchangeWithAgentSig"
167     address agentAddr = getSigner(coinsToMaker, coinsToTaker, uint(uint160(makerAddr)),
168     takerAddr_dueTime80_v8, r, s);
169     require(makerToAgent[makerAddr] == agentAddr, "invalid agent");
170   }
171   uint dueTime = uint80(takerAddr_dueTime80_v8>>8);
172   uint currTime = block.timestamp*MUL;
173   require(currTime < dueTime, "too late");
174   ...
175 }
```

Listing 2.3: ExchangeHub.sol

Impact N/A

Suggestion Move the check of `dueTime` and `currTime` before `getSigner`.

2.3.2 Ensure that the exchange function will not be abused

Status -

Description The `exchange` function is used by the contract in multiple scenarios, including hongbao [1], P2P order [2] and donation [3]. The basic idea is that the `maker` signs a transaction offline and passes the signed message to `taker`. The `take` will use this signed transaction to invoke the `exchange` function to transfer tokens from `maker` to `taker` and/or vice verse, depending on the parameters. For instance, if the `coinsToMaker` is not zero, the the `taker` will transfers the tokens specified in `coinsToMaker` to `maker`.

This mechanism works in general. However, we think it can be abused to launch the phishing attack. For instance, in the P2P order, the `maker` can put a fake token (which disguises as a valuable token) in the `coinsToTaker`, but put a valuable token in the `coinsToMaker`. If the `taker` executes the signed message on the chain, the `taker` will get the fake token but transfers the valuable token to the `maker`.

We understand that this issue is not solely caused by the smart contract. However, the DApp should have the mechanism such as whitelist or verification to the parameters (of this function call or on the DApp web interface) to mitigate the threat.

[1] https://brucelee.cash/2022/01/08/hongbaoclick/

[2] https://brucelee.cash/2022/01/15/p2p-orders/

[3] https://brucelee.cash/2022/01/17/starter-money/

Impact N/A

Suggestion -