BlockSec

Security Audit Report for Golff Bridge

Date: Aug 28, 2021

Version: 1.0

Contact: contract@blocksecteam.com

Contents

Report Manifest

Item	Description
Client	Golff Finance
Target	Golff Bridge
Auditors	Lei Wu, Yajin Zhou
Approved By	Lei Wu

Version History

Version	Date	Description
1.0	August 28, 2021	First Release

1 Introduction

1.1 About Golff Bridge

The Golff Bridge is a bridge for exchanging assets between the source chain and the target chain. The project consists of the smart contract (covered in this audit) and the relay daemon (out of the scope of this audit). The external function `bridgeIn` is used to accept user's assets from the source chain and then generate an event. The event will be captured by the relay (a service daemon). The daemon then invokes the `bridgeOut` to transfer the assets to the target chain.

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

Note that this audit is only for the smart contract of the bridge. **The relay is out of the scope of this audit**. The MD5 values of the files before and after the audit are shown in the following.

Before

File	MD5
BridgeChain.sol	`2f3e1afca08524d189921bbf236e6c12`
lib/ERC20Lib.sol	`1a67d7b72240580c53ceac5fcca1959c`

After

File	MD5
BridgeChain.sol	`6623d5e5c5ff9d281dcf985808377299`
lib/ERC20Lib.sol	`1a67d7b72240580c53ceac5fcca1959c`

1.2 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. The team won first place in the 2019 iDash competition (SGX Track). They can be reached at Email, Twitter and Medium.

1.3 Disclaimer

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. Besides, this report does not constitute personal investment advice or a personal recommendation.

1.4 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.

1.4.1 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

1.4.2 DeFi Security

- Semantic consistency
- Functionality consistency
- Access control

- Business logic
- Token operation
- Emergency mechanism
- Oracle security
- Whitelist and blacklist
- Economic impact
- Batch transfer

1.4.3 NFT Security

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

1.4.4 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.*

1.5 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/

2 Findings

In total, we have identified **4 potential issues**, as follows:

- High Risk: 0
- Medium Risk: 0
- Low Risk: 4

ID	Severity	Description	Category
1	Low	missed modifier	Software Security
2	Low	incorrect required condition in `bridgeBack`	Software Security
3	Low	misleading log information	Additional Recommendation
4	Low	unnecessary payable receive function	Additional Recommendation

The details are provided in the following sections.

2.1 Software Security

2.1.1 Missed modifier

Status Confirmed and fixed.

Description In the `recharge` function, there should be a `whenNotPaused` modifier.

```
function recharge(address _token, uint256 _amount) external payable {
  IERC20(_token).universalTransferFrom(msg.sender, address(this), _amount);
  Assets storage assets = assetsMapping[_token];
  assets.totalRecharge = assets.totalRecharge.add(_amount);
  emit Recharge(msg.sender, _token, _amount);
  }
```

Impact The function can still be invoked when the contract is paused.

Suggestion Add the `whenNotPaused` modifier.

2.1.2 Incorrect required condition in `bridgeBack`

Status Confirmed and fixed.

Description In the `bridgeBack` function, when the token is the base token (line 135), the balance should be bigger than $\_amount + \_fee$.

```
125 function bridgeBack(address _receiver, address _token, uint256 _amount, uint256
        _fee, string calldata _fromTxid) external onlyManager{
126   require(!backMapping[_fromTxid], 'BridgeChain:: txid returned');
```

```

    uint256 feeBalance = IERC20(feeToken).universalBalanceOf(address(this));
    require(feeBalance >= _fee, 'BridgeChain: greater than current balance');

    uint256 balance = IERC20(_token).universalBalanceOf(address(this));
    //This required condition is not correct when the token is the base token.
    require(balance >= _amount, 'BridgeChain: greater than current balance');

    if(IERC20(_token).isBase()){
      IERC20(_token).universalTransfer(_receiver, _amount.add(_fee));
    }else{

      IERC20(feeToken).universalTransfer(_receiver, _fee);

      IERC20(_token).universalTransfer(_receiver, _amount);
    }

    Fee storage fee = feeMapping[_token];
    fee.totalFee = fee.totalFee.sub(_fee);
    fee.currentFee = fee.currentFee.sub(_fee);

    uint256 record = recordMapping[_receiver][_token];
    require(record >= _amount, 'BridgeChain: greater than current record');
    recordMapping[_receiver][_token] = record.sub(_amount);
    backMapping[_fromTxid] = true;
    emit BridgeBack(msg.sender, _receiver, _token, _amount, _fee, _fromTxid);
    }
```

Impact The bridgeBack operation could fail due to insufficient balance.

Suggestion Ensure that $balance >= \_amount.add(\_fee)$ when the token is a base token.

2.2 Additional Recommendation

2.2.1 misleading log information

Status Confirmed and fixed.

Description In the `withdraw` function, the log information on line 171 should be “admin is the zero address".

```
170 function withdraw(address _receiver, address _token, uint256 _amount) external
        onlyAdmin {
171   require(_receiver != address(0), "BridgeChain: manager is the zero address");
172   uint256 balance = IERC20(_token).universalBalanceOf(address(this));
173   require(balance >= _amount, 'BridgeChain: greater than current balance');
174   IERC20(_token).universalTransfer(_receiver, _amount);
```

```
    Assets storage assets = assetsMapping[_token];
    assets.totalWithdraw = assets.totalWithdraw.add(_amount);
    emit Withdraw(msg.sender, _receiver, _token, _amount);
    }
```

Impact The wrong log information could cause confusion.

Suggestion Change "manager is the zero address" to "admin is the zero address"

2.2.2 unnecessary payable receive function

Status Confirmed and fixed.

Description There is an empty and payable `receive` function. This function is unnecessary since the function `bridgeIn` is for the purpose of receiving the token.

```
receive() external payable {}
```

Impact With this function, the token could be accidentally transferred to the contract, without going through the `bridgeIn` or `recharge` function.

Suggestion Remove this function.

2.3 Other Recommendations

2.3.1 The security of the relay

The relay is critical for the security of the cross-chain project. Since this audit is only for the smart contract, we recommend that the project owner should pay attention to the relay as well.

First, the security of the machine that runs the relay should be guaranteed. That's because if the relay is compromised then the relay can directly invoke the functions in the smart contract to transfer the tokens to arbitrary addresses.

Second, there is no acknowledgement mechanism to notify the source chain that the bridgeOut operation is successful. This causes the state synchronization challenge between source and target chains. The relay needs a mechanism to sync the states.

Third, when performing the bridgeOut operation, the relay needs to have the mechanisms to deal with timeout transactions and replace pending transactions if necessary.

2.3.2 The risk of the centralized design

This project has a highly centralized design. The admin and manager of the contract can withdraw the tokens (and fee) inside the contract. The project owner should enforce security mechanisms to protect the private keys of the contract deployer, admin and manager.

3 Conclusion

In this audit, we have analyzed the business logic, the design and the implementation of the Golff Bridge. Overall, the current code base is well structured and implemented, i.e., most of the identified issues have been promptly discussed, confirmed or fixed. Meanwhile, as previously disclaimed, this report does not give any warranties on discovering all security issues of the smart contracts. We appreciate any constructive feedback or suggestions.