Security Audit Report for Spherium Bridge Smart Contract

Date: January 17, 2022

Version: 1.1

Contact: contact@blocksecteam.com

BlockSec Audit Team

Digitally signed by BlockSec Audit Team
Date: 2022.01.17 18:41:40 +08'00'

Contents

Report Manifest

Item	Description
Client	Spherium
Target	Spherium Bridge Smart Contract

Version History

Version	Date	Description
1.0	December 24, 2021	First Release
1.1	January 17, 2022	Add the feedback from the project

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 MD5 values of the files before and after the audit are shown in the following.

Before

File	md5
WRAPPER.sol	0d3d73365cd4d741df39b7ed2453c08b

After

File	md5
WRAPPER.sol	544b262d98646664f7e8006b61e12c29

Note that, only WRAPPER.sol was audited. Other files were not included in the audit.

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 did not find issues that affect the smart contract. However, we do have two recommendations, as follows:

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

ID	Severity	Description	Category	Status
1	Medium	*The tokenName is not removed in* `removeTokenFromWhitelist`	DeFi Security	Confirmed & Fixed
2	Medium	*Insufficient check whether the tokenName is duplicated when adding to the whitelist*	DeFi Security	Confirmed & Fixed
3	Medium	*The token bridge fee of the token being removed from the whitelist will be locked*	DeFi Security	Confirmed & Fixed
4	-	*Be aware of the elastic supply tokens*	Recommendation	-
5	-	*Misc ones*	Recommendation	-

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 The tokenName is not removed in `removeTokenFromWhitelist`

Status Confirmed and fixed

Description

When removing a token from the whitelist (`removeTokenFromWhitelist`) , it does not remove the token name from `whitelistedTokenNames`. This can cause a failure when invoking the `claimAllTokenBridgeFee`, since the `tokenAddress` returned on Lin 452 could be 0.

```
283 function whitelistToken(address tokenAddress, string memory tokenName)
284 external
285 onlyOwner
286 returns (bool)
287 {
288   require(tokenAddress != address(0), "Cannot be address 0");
289   isWhitelisted[tokenAddress] = true;
290   whitelistedTokenName[tokenAddress] = tokenName;
291   whitelistedTokenAddress[tokenName] = tokenAddress;
292   whitelistedTokenNames.push(tokenName);
293   return true;
294 }
295
296 function removeTokenFromWhitelist(address tokenAddress)
297 external
```

```
onlyOwner
returns (bool)
{
  require(tokenAddress != address(0), "Cannot be address 0");
  string memory tokenName = whitelistedTokenName[tokenAddress];
  delete whitelistedTokenAddress[tokenName];
  delete whitelistedTokenName[tokenAddress];
  isWhitelisted[tokenAddress] = false;
  return true;
}
```

Listing 2.1: WRAPPER.sol

```
function claimAllTokenBridgeFee(address receiver)
external
onlyOwner
returns (bool)
{
  require(receiver != address(0), "Cannot be address 0");
  for (uint256 i = 0; i < whitelistedTokenNames.length; i++) {
    address tokenAddress = whitelistedTokenAddress[
    whitelistedTokenNames[i]
    ];
    uint256 fee = bridgeFee[tokenAddress];
    bridgeFee[tokenAddress] = 0;
    require(
    IERC20(tokenAddress).transfer(receiver, fee),
    "There was a problem transferring your tokens"
    );
  }

return true;
}
```

Listing 2.2: WRAPPER.sol

Impact This can cause a failure when invoking the `claimAllTokenBridgeFee`, making the token bridge fee cannot be claimed.

Suggestion Remove the token name from `whitelistedTokenNames`.

2.1.2 Insufficient check whether the tokenName is duplicated when adding to the whitelist

Status Confirmed and fixed

Description When adding token address and token name to the white list (`whitelistToken`), it does not check whether the added token name is duplicated. That's because different token addresses can have the same token name. In this case, if the added token name exists in the white list but has a different token address, then the newly added token name will overwrite the old ones in `whitelistedTokenAddress` (line 291). This can further cause the problem when claiming the token bridge fee (`claimAllTokenBridgeFee`), since it looks up the `whitelistedTokenAddress` to get the corresponding token address.

```
function whitelistToken(address tokenAddress, string memory tokenName)
external
onlyOwner
returns (bool)
{
  require(tokenAddress != address(0), "Cannot be address 0");
  isWhitelisted[tokenAddress] = true;
  whitelistedTokenName[tokenAddress] = tokenName;
  whitelistedTokenAddress[tokenName] = tokenAddress;
  whitelistedTokenNames.push(tokenName);
  return true;
}
```

Listing 2.3: WRAPPER.sol

Impact The newly added token name will overwrite the old one in whitelistedTokenAddress. This can make that the token bridge fee can be claimed for the old token.

Suggestion Check whether the token name exists when adding to the white list.

2.1.3 The token bridge fee of the token being removed from the whitelist will be locked

Status Confirmed and fixed

Description When removing a token from the whitelist, it does not claim the remaining token bridge fee. This can cause the bridge fee for that token freeze in the pool since the function `claimAllTokenBridgeFee` only claims the bridge fee for the tokens in the white list.

Impact This can cause the bridge fee for the token that has been removed from the whitelist freeze in the pool.

Suggestion Claim the bridge fee for the token when removing it from the white list.

2.2 Additional Recommendation

2.2.1 Be aware of the elastic supply tokens

Description Elastic supply tokens could dynamically adjust their price, supply, user's balance, etc. Such a mechanism makes a DeFi system complex, while many security accidents are caused by the elastic tokens. As a centralized bridge, however, the invocation of the deposit/withdraw functions may fail due to the elastic supply tokens. The tokens added to the whitelist should be carefully verified.

2.2.2 Misc ones

Description This bridge is centralized. Many of the functions can only be invoked by the owner of the contract. The project should pay attention to securing the private key of the contract owner, since may incidents are caused by the private key leakage. A multi-signature wallet could be leveraged, and MPC and TEE based private key protection could be used.

Feedback from the project The development team is currently adding the support to make the contract owner signature as a multisig one.