Security Audit Report for FBTC & Fire Bridge Contracts

Date: June 20, 2024 **Version:** 1.1

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Fire Bitcoin
Target	FBTC & Fire Bridge Contracts

Version History

Version	Date	Description
1.0	May 9, 2024	First release
1.1	June 20, 2024	Update commit hash

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2024-06-21 06:32:36+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec 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 successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. 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 focus of this audit is the FBTC & Fire Bridge Contracts [1] of the Fire Bitcoin project. These smart contracts serve as a wrapper for the Bitcoin cryptocurrency (BTC), featuring integrated cross-chain functionality. It is important to note that all dependencies of the smart contracts within the audit scope are considered reliable in both functionality and security. Therefore, they are excluded from the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
FBTC & Fire Bridge Contracts	Version 1	0013f3aa84a38a61824c7338298378ea498151b0
	Version 2	cc0cadecd35b75b3a71a24017e03838ffde18e7f

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 does 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] https://github.com/fbtc-xyz/fbtc-contract

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
* Permission management
* 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]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

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

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item 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 found **three** potential security issues. Besides, we have **one** note.

- Low Risk: 3
- Note: 1

ID	Severity	Description	Category	Status
1	Low	Potential front-running DoS attack	Software Security	Fixed
2	Low	Lack of checking for key parameters	Software Security	Fixed
3	Low	Lack of verification for the target chain in cross-chain requests	Software Security	Fixed
4	-	Potential centralization risks	Note	-

The details are provided in the following sections.

2.1 Software Security

2.1.1 Potential front-running DoS attack

Severity Low

Status Fixed in Version 2

Introduced by Version 1

Description The FireBridge contract is in charge of minting and burning the FBTC tokens corresponding to operations on the Bitcoin chain. During the minting process, a qualified user must first call the addMintRequest function with the ID and output index of the deposit transaction on the Bitcoin chain. This function marks the transaction as *used*, preventing further usage.

However, a potential front-running DoS attack exists, where another qualified user is able to front-run the call to the addMintRequest function and use the same deposit transaction. Since the FireBridge contract cannot validate the deposit transaction, the front-run may succeed, effectively preventing the legitimate user from using the deposit transaction.

```
function addMintRequest(
    uint256 _amount,
    bytes32 _depositTxid,
    uint256 _outputIndex
)
    external
    onlyQualifiedUser
    whenNotPaused
    returns (bytes32 _hash, Request memory _r)
{
    // Check request.
    require(_amount > 0, "Invalid amount");
    require(uint256(_depositTxid) != 0, "Empty deposit txid");
    bytes memory _depositTxData = abi.encode(_depositTxid, _outputIndex);
```

```

        bytes32 depositDataHash = keccak256(_depositTxData);
        require(
            usedDepositTxs[depositDataHash] == bytes32(uint256(0)),
            "Used BTC deposit tx"
        );

        // Compose request. Main -> Self
        _r = Request({
            nonce: nonce(),
            op: Operation.Mint,
            srcChain: MAIN_CHAIN,
            srcAddress: bytes(depositAddresses[msg.sender]),
            dstChain: chain(),
            dstAddress: abi.encode(msg.sender),
            amount: _amount,
            fee: 0, // To be set in `_splitFeeAndUpdate`
            extra: _depositTxData,
            status: Status.Pending
        });

        // Split fee.
        _splitFeeAndUpdate(_r);

        // Save request.
        _hash = _addRequest(_r);

        // Update deposit data usage status.
        usedDepositTxs[depositDataHash] = _hash;
    }
```

Listing 2.1: FireBridge.sol

Impact A qualified user may be able to launch a front-running attack, potentially causing a Denial of Service (DoS) for other users.

Suggestion Revise the corresponding logic.

2.1.2 Lack of checking for key parameters

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `FireBridge` contract, there are no check for the validity of the new parameters when setting key parameters of the bridge. For example, there is no verification whether the new fee model address and the fee recipient address are zero addresses when setting these paramters. If these addresses are set to zero, the normal functionality of the `FireBridge` contract may fail. Specifically, if the fee recipient address is set to zero, then no users would be able to add mint, burn or cross-chain requests because OpenZeppelin's implementation of ERC-20 tokens bans direct transfer to the zero address.

The following is the code segment for the functions that set key paramters:

```
function setFeeModel(address _feeModel) external onlyOwner {
    feeModel = _feeModel;
    emit FeeModelSet(_feeModel);
}

function setFeeRecipient(address _feeRecipient) external onlyOwner {
    feeRecipient = _feeRecipient;
    emit FeeRecipientSet(_feeRecipient);
}
```

Listing 2.2: FireBridge.sol

Impact Mistakenly setting values may result in incorrect functionality of the contracts.

Suggestion Add sanity checks to the functions that set key paramters.

2.1.3 Lack of verification for the target chain in cross-chain requests

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `FireBridge` contract, the `addCrosschainRequest` function is used to initiate cross-chain requests for the `FBTC` token. However, it fails to check whether the `targetChain` is valid.

```
function addCrosschainRequest(
    bytes32 _targetChain,
    bytes calldata _targetAddress,
    uint256 _amount
) external whenNotPaused returns (bytes32 _hash, Request memory _r) {
```

Listing 2.3: FireBridge.sol

Impact Invalid target chain as the parameter may lead to unexpected results.

Suggestion Add sanity checks for the `targetChain` parameter.

Feedback from the Project There could be numerous possible values for the target chains. Implementing a check for the target chain would require updating the entire list whenever a new chain is supported. To ensure the correctness of the value, we will add strict checks in the front-end of the dApp. If users call the functions directly, it is their responsibility to verify the correctness of the target chain.

2.2 Note

2.2.1 Potential centralization risks

Description The `FBTC` token and the `FireBridge` contract pose potential centralization risks due to the following reasons:

- The entire cross-chain process relies on several privileged operations and accounts.
- There is a lack of explicit refund mechanism.
- The project maintainers have the ability to arbitrarily control the minting and burning mechanism indirectly.
- The syncing of blacklists and qualified users relies on the correct operation of the project maintainers.

BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS