Competitive Security Assessment

FireBitcoin_lockedFBTC

Jun 26th, 2024

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

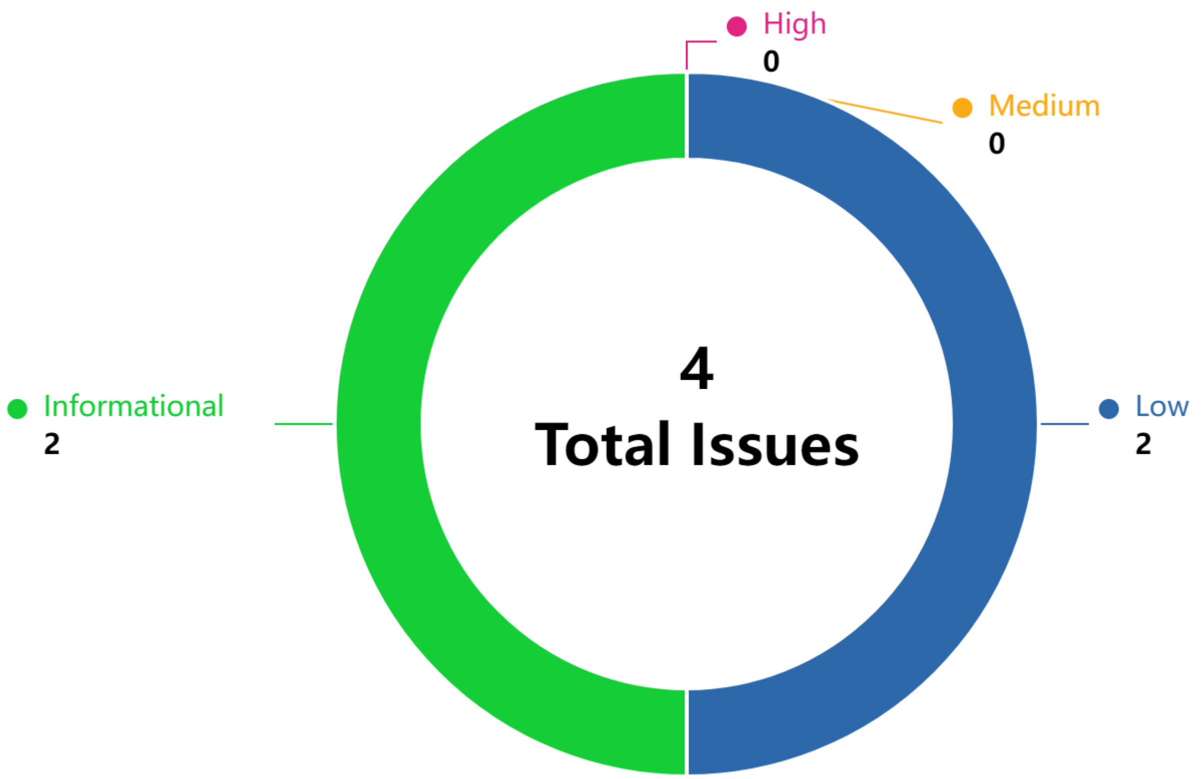
Overview

Project Name	FireBitcoin_lockedFBTC
Language	solidity
Codebase	• https://github.com/fbtc-com/fbtcX-contract • audit version - ae3a6ace8e073115e1e82a338951c925bf8e3988 • final version - a0c83ad5547ee006ad8e515972c4c09c16694ed8
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
src/LockedFBTC.sol	52b84db6224a2785fb4450edf803b5ac3bcb57a648a46cda2322ce069319313a
src/Common.sol	fcd2753db4bcdeed56cc92e9cc4d2cc9efef5a6b4874d9b55348b2122c5e0853
src/Interfaces/IFireBridge.sol	6ee05bf81ff5273d86b7f2832ba9855c96213378c75ad9c6ea9bb705014e2837

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
FBX-1	The `mintLockedFbtcRequest()` function allows zero minting, leading to unnecessary gas and fee expenditure	Logical	Low	Fixed	0xzoobi, 8olidity
FBX-2	Lack of check on returned value	Logical	Low	Fixed	biakia
FBX-3	Unused imports	Language Specific	Informational	Fixed	biakia
FBX-4	Missing zero address check	Logical	Informational	Fixed	Cara, biakia, Saaj

FBX-1:The `mintLockedFbtcRequest()` function allows zero minting, leading to unnecessary gas and fee expenditure

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	0xzoobi, 8olidity

Code Reference

- code/src/LockedFBTC.sol#L75

```
75: realAmount = _amount - _r.fee;
```

Description

0xzoobi: In the `mintLockedFbtcRequest()` function, there is an issue where `realAmount` could be zero if the `_amount` passed to the function is equal to the fee deducted by the `addBurnRequest` function of the `IFireBridge` contract. When this happens, the user ends up spending gas fees and the fee amount `_r.fee` without receiving any new FBTC tokens. This scenario leads to unnecessary expenditure for the user with no tangible benefit.
8olidity: This code defines a contract called `LockedFBTC` which is an ERC20 token contract that represents a locked version of the FBTC (Fireblocks Bitcoin) token. The `mintLockedFbtcRequest` function is used to mint new `lockedFBTC` tokens.
The function works as follows:

1. It checks that the input `_amount` is greater than zero.
2. It checks that the caller has a sufficient FBTC balance to cover the requested amount.
3. It transfers the requested amount of FBTC from the caller to the contract.
4. It calls the `addBurnRequest` function of the `IFireBridge` contract to create a new burn request. This burn request represents the locking of the FBTC tokens.
5. It calculates the "real amount" to be minted by subtracting the fee from the requested amount.
6. It mints the "real amount" of `lockedFBTC` tokens and transfers them to the caller.
7. It emits a `MintLockedFbtcRequest` event.

If the `_amount` passed to the function is equal to the `_r.fee` returned by the `addBurnRequest` function, then the "real amount" to be minted will be zero. This means that the caller will not receive any `lockedFBTC` tokens, despite having transferred FBTC to the contract.
poc

```
    function testRedeemFbtcRequestburn2() public {
        vm.startPrank(minter);
        fbtc0Mock.approve(address(lockedFBTC), 1 * 10 ** 4);
        lockedFBTC.mintLockedFbtcRequest(1 * 10 ** 4);
        assertTrue(lockedFBTC.balanceOf(minter) == 0);

}
```

Recommendation

0xzoobi: To prevent this, a check should be added to ensure that `realAmount` is greater than zero before proceeding with the minting and emitting events. If `realAmount` is zero, the function should revert with an appropriate error message.
Sample Fix:

```
function mintLockedFbtcRequest(uint256 _amount)
    public
    onlyRole(MINTER_ROLE)
    whenNotPaused
    returns (uint256 realAmount)
{

//some code

  realAmount = _amount - _r.fee;
require(realAmount > 0, "Real amount must be greater than zero after fee deduction.");

    _mint(msg.sender, realAmount);

}
```

8olidity: Modify the `mintLockedFbtcRequest` function to ensure that the "real amount" to be minted is always greater than zero, even if the requested amount is equal to the fee. This could be done by either:

1. Rejecting the request if the "real amount" would be zero.
2. Minting a minimum non-zero amount of `lockedFBTC` tokens, even if the "real amount" is zero.

Implementing one of these solutions will ensure that users always receive some `lockedFBTC` tokens when calling the `mintLockedFbtcRequest` function, as long as they have a sufficient FBTC balance.

Client Response

client response for 0xzoobi: Fixed. commit - a0c83ad5547ee006ad8e515972c4c09c16694ed8
client response for 8olidity: Fixed. commit - a0c83ad5547ee006ad8e515972c4c09c16694ed8

FBX-2:Lack of check on returned value

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	biakia

Code Reference

- code/src/LockedFBTC.sol#L81-L91

```
81: function redeemFbtcRequest(uint256 _amount, bytes32 _depositTxid, uint256 _outputIndex)
82:         public
83:         onlyRole(MINTER_ROLE)
84:         whenNotPaused
85:         returns (bytes32 _hash, Request memory _r)
86:     {
87:         require(_amount > 0 && _amount <= totalSupply(), "Amount out of limit.");
88: 
89:         (_hash, _r) = IFireBridge(fbtcBridge).addMintRequest(_amount, _depositTxid, _outputIndex);
90:         emit RedeemFbtcRequest(msg.sender, _depositTxid, _outputIndex, _amount);
91:     }
```

Description

biakia: The function `redeemFbtcRequest` does not check if the mint request was successful before emitting the `RedeemFbtcRequest` event:

```
(_hash, _r) = IFireBridge(fbtcBridge).addMintRequest(_amount, _depositTxid, _outputIndex);
```

This means that even if the mint request fails, the event will still be emitted, leading to a potential inconsistency between the emitted event and the actual state of the contract.

Recommendation

biakia: Consider following fix:

```
(_hash, _r) = IFireBridge(fbtcBridge).addMintRequest(_amount, _depositTxid, _outputIndex);
require(_hash != bytes32(uint256(0)), "Failed to create a valid mint request.");
```

Client Response

client response for biakia: Fixed. commit - a0c83ad5547ee006ad8e515972c4c09c16694ed8

FBX-3:Unused imports

Category	Severity	Client Response	Contributor
Language Specific	Informational	Fixed	biakia

Code Reference

- code/src/LockedFBTC.sol#L12

```
12: import {Request, UserInfo, RequestLib, Operation} from "./Common.sol";
```

Description

biakia: The imports `UserInfo` , `RequestLib` and `Operation` are not used in `LockedFBTC` .

Recommendation

biakia: Consider removing these unused imports.

Client Response

client response for biakia: Fixed. commit - a0c83ad5547ee006ad8e515972c4c09c16694ed8

FBX-4:Missing zero address check

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	Cara, biakia, Saaj

Code Reference

- code/src/LockedFBTC.sol#L31-L38
- code/src/LockedFBTC.sol#L31-L49
- code/src/LockedFBTC.sol#L31

```
31: function initialize(
32:         address _fbtcAddress,
33:         address _fbtcBridgeAddress,
34:         address admin,
35:         address pauser,
36:         address minter,
37:         address safetyCommittee
38:     ) public initializer {
```

```
31: function initialize(
32:         address _fbtcAddress,
33:         address _fbtcBridgeAddress,
34:         address admin,
35:         address pauser,
36:         address minter,
37:         address safetyCommittee
38:     ) public initializer {
39:         __ERC20_init("lockedFBTC", "lockedFBTC");
40:         __Pausable_init();
41:
42:         _grantRole(DEFAULT_ADMIN_ROLE, admin);
43:         _grantRole(PAUSER_ROLE, pauser);
44:         _grantRole(MINTER_ROLE, minter);
45:         _grantRole(SAFETY_COMMITTEE_ROLE, safetyCommittee);
46:
47:         fbtcBridge = IFireBridge(_fbtcBridgeAddress);
48:         fbtc = IERC20Upgradeable(_fbtcAddress);
49:     }
```

```
31: function initialize(
```

Description

Cara: The input parameter of the address type in the function does not use the zero address for verification.
biakia: The function `initialize` lacks of zero address check on `_fbtcBridgeAddress` and `_fbtcAddress`.
Saaj: ## Vulnerability Detail
Contract `LockedFBTC` lack address(0) in its `initialize` function which can lead to address zero being set for all crucial addresses i.e.; `_fbtcAddress`, `_fbtcBridgeAddress`, `admin`, `pauser`, `minter` and `safetyCommittee`.

Impact

`initialize` function is a vital function in `LockedFBTC` contract which is used to set address for all important roles and contracts that are vital for `FBTC` contract.
If any of the contract or role is set to `address(0)` there will be no option than redeploying the contract again.

Recommendation

Cara: It is recommended to perform zero address verification on the input parameters of the address type.
biakia: Consider following fix:

```
function initialize(
        address _fbtcAddress,
        address _fbtcBridgeAddress,
        address admin,
        address pauser,
        address minter,
        address safetyCommittee
    ) public initializer {
        __ERC20_init("lockedFBTC", "lockedFBTC");
        __Pausable_init();

        _grantRole(DEFAULT_ADMIN_ROLE, admin);
        _grantRole(PAUSER_ROLE, pauser);
        _grantRole(MINTER_ROLE, minter);
        _grantRole(SAFETY_COMMITTEE_ROLE, safetyCommittee);
        require(_fbtcBridgeAddress!=address(0),"zereo address");
        require(_fbtcAddress!=address(0),"zereo address");
        fbtcBridge = IFireBridge(_fbtcBridgeAddress);
        fbtc = IERC20Upgradeable(_fbtcAddress);
    }
```

Saaj: The recommendation is made for implementing address zero check in `initialize` function for `LockedFBTC` contract to avoid setting roles to address(0) due to its vital importance.

```
function initialize(
        address _fbtcAddress,
        address _fbtcBridgeAddress,
        address admin,
        address pauser,
        address minter,
        address safetyCommittee
    ) public initializer {
+        require(admin != address(0), "Admin cannot be zero Address");
        __ERC20_init("lockedFBTC", "lockedFBTC");
        __Pausable_init();
        _grantRole(DEFAULT_ADMIN_ROLE, admin);
        _grantRole(PAUSER_ROLE, pauser);
        _grantRole(MINTER_ROLE, minter);
        _grantRole(SAFETY_COMMITTEE_ROLE, safetyCommittee);

        fbtcBridge = IFireBridge(_fbtcBridgeAddress);
        fbtc = IERC20Upgradeable(_fbtcAddress);
    }
```

Client Response

client response for Cara: Fixed. commit - a0c83ad5547ee006ad8e515972c4c09c16694ed8
client response for biakia: Fixed. commit - a0c83ad5547ee006ad8e515972c4c09c16694ed8
client response for Saaj: Fixed. commit - a0c83ad5547ee006ad8e515972c4c09c16694ed8

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.