FIREBTC FBTC SECURITY AUDIT REPORT

Jun 26, 2024

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals

- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

Fire Bitcoin (FBTC) is a Bitcoin-pegged token enabling cross-chain transfers between Bitcoin and EVM-compatible blockchains using a bridging mechanism facilitated by the FireBridge and FBTCMinter smart contracts. While users interact with FireBridge to initiate mint and burn requests, confirmations are delegated to assigned operators for various actions. Although this operator model offers flexibility and aims for enhanced security, the system currently relies on a centralized qualification process controlled by the FireBridge contract owner, limiting user participation.

1.4 Project Dashboard

Project Summary

Title	Description
Client	FireBTC
Project name	FBTC
Timeline	09.05.2024 - 25.06.2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
09.05.2024	d557e5e0b73eda5af86de5bf4431653056844e64	Initial commit
17.06.2024	cc0cadecd35b75b3a71a24017e03838ffde18e7f	Commit for the re-audit

Project Scope

The audit covered the following files:

File name	Link
contracts/FireBridge.sol	FireBridge.sol
contracts/Minter.sol	Minter.sol
contracts/FToken.sol	FToken.sol
contracts/FBTC.sol	FBTC.sol

File name	Link
contracts/Governable.sol	Governable.sol
contracts/Common.sol	Common.sol
contracts/FeeModel.sol	FeeModel.sol
contracts/FBTCGovernorModule.sol	FBTCGovernorModule.sol
contracts/BridgeStorage.sol	BridgeStorage.sol

Deployments

File name	Contract deployed on mainnet	Comment
FBTCMinter	0x80b534...0634AED7	
FBTC	0xC96dE2...3BD6C364	
FeeModel	0xd12D39...8841294e	
FireBridge	0xbee335...11395943	proxy
FireBridge	0xc5e2f8...8ed6f6e5	implementation
FBTCGovernorModule	0x0d7718...38914547	

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	0
Medium	3
Low	7

ID	Name	Severity	Status
M-1	Centralization risks	Medium	Fixed
M-2	Lack of chains whitelisting	Medium	Fixed
M-3	The `_getFee` function is not monotonic	Medium	Acknowledged
L-1	Zero address `feeRecipient` risk	Low	Fixed
L-2	Invalid parameter handling in view functions	Low	Fixed
L-3	`_disableInitializers()` can be used	Low	Acknowledged
L-4	Different ownable contracts are used (`Ownable`, `Ownable2Step`)	Low	Fixed
L-5	The getFee function does not raise an exception if the configuration is not defined.	Low	Fixed
L-6	Misleading range / incorrect error message in `getRequestsByIdRange`	Low	Fixed
L-7	Lack of validation for the zero-width fee tiers	Low	Acknowledged

1.6 Conclusion

This audit focused exclusively on the EVM component of the FBTC bridge, examining the Solidity smart contracts responsible for minting, burning, and cross-chain transfer of FBTC tokens. Our analysis included a high-level review of the backend architecture as detailed in the provided whitepaper.

FBTC aims to maintain a 1:1 peg with BTC, achieved by locking BTC in custodial addresses on the Bitcoin blockchain. Authorized users, having completed KYC/AML procedures, can initiate FBTC minting on EVM-compatible chains by sending BTC to these custodial addresses. The `FireBridge` contract facilitates this process, relying on confirmations from a network of Threshold Signature Scheme (TSS) Nodes to validate BTC transactions and mint corresponding FBTC tokens. Non-authorized users can transfer FBTC cross-chain using a similar mechanism, with confirmations from TSS Nodes.

The primary attack vectors and considerations examined during the audit are outlined below:

1. **Centralization risks**:

    - The TSS Network consists of multiple independent projects that individually confirm and sign cross-chain requests, as stated in the whitepaper. However, the `FireBridge` and `FBTC` contracts are upgradeable proxies controlled by a single owner, a multi-sig Gnosis Safe account. This centralization places significant trust in the owner and holders of crucial roles such as `USER_MANAGER`, `LOCKER` and `FBTCMinter`, who are responsible for configuration of authorized users, censorship, and cross-chain operations.
    - The `burn` and `payFee` functions of the `FToken` contract allow the bridge contract to burn or transfer tokens from any address without requiring a spending allowance. This could potentially create an attack vector with a compromised bridge contract to drain users' `FBTC` balances.
    - To enhance trustlessness and security, it is recommended to transition to a DAO approach for ownership and further decentralize the TSS Gateway.

2. **Cross-Chain messaging confirmations**:

    - Each TSS Node is responsible for implementing its own checks to ensure:
        - The transaction on the sender chain should be included in the finalized block, preventing rollbacks after TSS Node confirmation. Special consideration is needed for L2 networks, as they can use different finalization approaches, such as finalization of the transaction with a zk proof validation included into L1 network for zk rollups or after a challenge period for optimistic rollups. Failing to meet finalization requirements increases the risk of fraudulent activity targeting the least secured network.
        - The BTC-EVM chain and EVM chain-BTC gateways are partially validated by the `FireBridge` contract to prevent replay attacks by disallowing confirmation of requests with already confirmed transaction ID and txout. However, validating the BTC transaction, verifying the depositor or withdrawal address of authorized users, and conducting comprehensive AML checks need to be performed by the TSS Nodes.

3. **Incomplete test coverage**:

    - The project lacks tests for negative cases where transactions should revert on invalid addition or confirmation of requests. Additionally, the `FBTCGovernorModule` is untested. Comprehensive testing is crucial for continuous development to ensure that code improvements do not unintentionally violate the invariants of previous versions. It is recommended to add unit tests that fully cover all reachable branches of the codebase and include fuzzing tests to confirm the system's robustness.

4. **Lack of whitelisting**:

    - The `FireBridge.addCrosschainRequest()` and `confirmCrosschainRequest()` functions do not verify whether `_targetChain` and `srcChain` are whitelisted. This oversight allows cross-chain interactions with unsupported chains, increasing the risk of inadvertently burning users' tokens by sending them to unsupported chains or processing nonexistent confirmations of cross-chain requests by malicious TSS Gateways.

Additional suggestions for improvement

- **Operational monitoring**: Establishing real-time monitoring and alerting mechanisms for unusual activities can help detect and respond to potential security incidents promptly.
- **Documentation and transparency**: Providing comprehensive and up-to-date documentation for all smart contract functionalities and system operations can improve transparency and facilitate audits and reviews.
- **Periodic audits**: Conducting regular security audits and code reviews can ensure ongoing security and identify potential vulnerabilities as the system evolves.

By addressing these recommendations and continuously improving the system's security and transparency, the project can enhance its robustness and trustworthiness, providing a safer and more reliable platform for its users.

2. FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

M-1	Centralization risks
Severity	Medium
Status	Fixed in 33737b45

Description

The smart contract system of the bridge exhibits significant centralization and lack of trustlessness. The owner has the authority to change the implementation of both the `FireBridge` and `FToken` contracts.

Minting of FBTC on the EVM chains, cross-chain EVM transfers, and spending the BTC locked at the Bitcoin-EVM bridge are also centralized.

Additionally, the owner can set unlimited fees, as the `FeeConfig.minFee` parameter is not capped.

Related code: FeeModel.sol#L39

Recommendation

While centralization and trustfulness may be intended by design, it is recommended to implement constraints to cap the `minFee` parameter to prevent the setting of unlimited fees. This can help mitigate risks associated with excessive centralization and enhance the trustworthiness of the system.

Additionally, we recommend improving the validation infrastructure to bring more decentralization to the minting and spending process.

Client's Commentary

> `minFee` capped in the fix. Furthermore the owner will be Safe MultiSig wallet and will transfer to timelock contract or DAO contract in the future.

M-2	Lack of chains whitelisting
Severity	Medium
Status	Fixed in 9de76912

Description

The project provides for cross-chain token transfers of FBTC through the `addCrosschainRequest` function. If the network with the specified chain ID is not supported by the FBTC infrastructure, all burned FBTC tokens in the `addCrosschainRequest` function will be lost.

This issue is classified as MEDIUM severity since it occurs only if the user specifies an incorrect network or if the project's website frontend misleads the user.

Related code: FireBridge.sol#L308

Recommendation

We recommend adding a chain whitelisting in the `addCrosschainRequest`() function.

M-3	The `_getFee` function is not monotonic
Severity	Medium
Status	Acknowledged

Description

According to the developer's statement, the `_getFee` function is intended to be monotonic. However, the current implementation might not be monotonic if several fee tiers are configured. This issue cannot be corrected by any configuration due to the nature of the current implementation of the `_getFee` function.

Recommendation

We recommend adjusting the implementation of the `_getFee` function to comply with the intended behavior.

Client's Commentary

> We want to encourage mint&burn large numbers of FBTC. We acknowledge this issue.

2.4 Low

L-1	Zero address `feeRecipient` risk
Severity	Low
Status	Fixed in 2e052689

Description

If the `feeRecipient` is unset, the `confirmMintRequest` function triggered by the minter fails due to the transfer of fees to the zero address. Meanwhile, the `addMintRequest` function, triggered by the user before the confirmation, completes successfully. This situation partially blocks the minting requests by the gateway until the `feeRecipient` is set to a non-zero address.

Related code: FireBridge.sol#L77

Recommendation

We recommend adding a check to ensure that the `feeRecipient` is not a zero address in the `confirmMintRequest` function, or explicitly initialize it in the `initialize`() function.

L-2	Invalid parameter handling in view functions
Severity	Low
Status	Fixed in 33737b45

Description

The view functions `getRequestById` and `getRequestsByIdRange` do not check that the `id` and `start` parameters, respectively, are greater than 0. This oversight is problematic because the first element of the `requests` array is a dummy struct.

Related code: FireBridge.sol#L567

Recommendation

We recommend adding checks to ensure that the `id` and `start` parameters are greater than 0 in the `getRequestById` and `getRequestsByIdRange` functions, respectively.

L-3	`_disableInitializers()` can be used
Severity	Low
Status	Acknowledged

Description

All initialize calls in the implementation contract should be blocked to protect against various attacks. This can be done by calling initialize in the constructor (as currently implemented), but the most efficient method is to use the `_disableInitializers()` function (Initializable.sol#L192).

Recommendation

We recommend using the `_disableInitializers()` function to block initialize calls in the implementation.

Client's Commentary

> Instead of `_disableInitializers()` we called `initialize` in the constructor to block re-initialization.

L-4	Different ownable contracts are used (`Ownable`, `Ownable2Step`)
Severity	Low
Status	Fixed in 33737b45

Description

The project uses different contracts for ownership management. `FBTCMinter` and `FBTCGovernorModule` inherit from the `Ownable` contract, while FBTC and FireBridge inherit from Governable, which uses `Ownable2Step`. `Ownable2Step` inherits from `Ownable` and overrides the `transferOwnership`() method to make the new owner "pending." The recipient must then call `acceptOwnership`() to complete the transfer. This ensures that only an address with access to its private keys or control over the smart contract address can manage the smart contract.

Related code: RoleBasedAccessControl.sol#L10

Recommendation

We recommend using Ownable2Step in all contracts as it provides a more secure ownership transfer mechanism.

L-5	The getFee function does not raise an exception if the configuration is not defined.
Severity	Low
Status	Fixed in 33737b45

Description

The `getFee` function is intended to return a fee based on the amount, operation type, and chain. The chain-specific fee will be returned if configured; otherwise, the default fee for the given operation will be returned. If neither a chain-specific nor a default fee is configured, a zero value will be returned.

It appears that the case of no fee configured is a configuration error, leading to unintended operations with a zero fee value.

Related code: FeeModel.sol#L87

Recommendation

We recommend reverting the transaction if no fees are configured. A zero fee value, if intended, can be explicitly configured by setting the zero value in the corresponding configuration.

L-6	Misleading range / incorrect error message in `getRequestsByIdRange`
Severity	Low
Status	Fixed in 33737b45

Description

If the caller specifies `start` = `end`, which may be intention to obtain exactly one item, they receive the `start > end` error, which is misleading and incorrect.

Related code: FireBridge.sol#L573

Recommendation

We recommend correcting the error message or changing the meaning of the `end` value to be the exact upper range.

L-7	Lack of validation for the zero-width fee tiers
Severity	Low
Status	Acknowledged

Description

Due to the FeeModel.sol#L93, the `_validateConfig` function allows subsequent fee tiers to have the same boundaries. As the fee tiers are configured by a privileged user, such a configuration is unlikely to be set up. Additionally, no contract malfunction will occur under such conditions. However, subsequent fee tiers with the same boundaries are meaningless and not intended, so it would be beneficial if the validation procedure disallowed it.

Recommendation

We recommend disallowing subsequent fee tiers with the same boundaries by changing the non-strict condition to a strict one.

Client's Commentary

> The owner will ensure the fee tiers is correct and reasonable thus the contract does not intend to perform perfectly strict check.

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://twitter.com/mixbytes