Three Sigma

Code Audit

ORANGE

Orange Bridge Cross chain Bitcoin interoperability

Disclaimer

Code Audit

Orange Bridge Cross chain Bitcoin interoperability

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

Orange Bridge Cross chain Bitcoin interoperability

Table of Contents

Summary

Code Audit

Orange Bridge Cross chain Bitcoin interoperability

Summary

Three Sigma audited Orange Bridge in a 3 day engagement. The audit was conducted from 15-07-2024 to 17-07-2024.

Protocol Description

Orange Bridge is a cross-chain bridge that enables interoperability across the Bitcoin ecosystem and other blockchain networks. It effortlessly bridges tokens back and forth between the Bitcoin BRC20 and Ethereum ERC20 protocols.

Components:

- Orange Bridge Web DApp - The user interface that facilitates user interactions and blockchain services.
- Smart Contract - Manages the burning and minting of ERC20 tokens.
- EVM Event Listener - Listens for events on the Ethereum Virtual Machine (EVM) chain.
- Bitcoin BRC20 Event Listener - Listens for events on the Bitcoin chain related to BRC20 tokens.
- ERC20 Token Sending Service - Handles the sending of ERC20 tokens.
- BRC20 Token Sending Service - Manages the sending of BRC20 tokens.

Scope

Code Audit

Orange Bridge Cross chain Bitcoin interoperability

Scope

Components
orange-bridge-contract

Assumptions

External libraries are safe.

Methodology

Code Audit

Orange Bridge Cross chain Bitcoin interoperability

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

Orange Bridge Cross chain Bitcoin interoperability

Project Dashboard

Application Summary

Name	Orange Bridge
Commit	cdc8ac673425e19909639fd6c689ab2635d7b8bd
Language	Solidity
Platform	Ethereum

Engagement Summary

Timeline	15-07-2024 to 17-07-2024
№ of Auditors	1
Review Time	3 days

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	3	3	0
Low	7	6	0

None	7	6	1

Category Breakdown

Suggestion	7
Documentation	0
Bug	5
Optimization	5
Good Code Practices	0

Code Maturity Evaluation

Code Audit

Orange Bridge Cross chain Bitcoin interoperability

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. The codebase has a strong access control mechanism.
Arithmetic	Satisfactory. The codebase uses Solidity version >0.8.0 implementing safe arithmetic.
Centralization	Weak. The admins/owners of the protocol have significant privileges over the protocol.
Code Stability	Satisfactory. The codebase was stable during the course of the audit.
Upgradeability	Satisfactory. Orange Bridge is upgradeable.
Function Composition	Satisfactory. Functionality is well split into different helpers.
Front-Running	Satisfactory. No front-running issues were identified.
Monitoring	Satisfactory. Most state changes correctly emit events.
Specification	Satisfactory. The code matched the design specifications.
Testing and Verification	Satisfactory. Unit and integration tests were present for most functionality.

Findings

Code Audit

Orange Bridge Cross chain Bitcoin interoperability

Findings

3S-OB-M01

Balance check in **`Vault::withdraw()`** does not take fees into account

Id	3S-OB-M01
Classification	Medium
Severity	High
Likelihood	Low
Category	Bug
Status	Addressed in #cdd25c9.

Description

`Vault::withdraw()` transfers funds to an user if **`address(this)).balance >= amount`**, which may transfer funds resulting from fees.

Recommendation

Replace the check by **`address(this)).balance - totalFees >= amount`** to make sure fees can always be withdrawn.

3S-OB-M02

`address.transfer` is used in the codebase, which could lead to stuck funds

Id	3S-OB-M02
Classification	Medium
Severity	High
Likelihood	Low
Category	Bug
Status	Addressed in #252ae5e.

Description

Using **`address.transfer`** is not recommended as future gas cost changes can make it no longer work or users might use smart contract wallets. The reason is that it only forwards **`2300`** gas, which can lead to the 2 problems mentioned before.

Recommendation

Use **`address.call{value: amount}("")`** instead, here is an example implementation. If the gas usage of the target is a concern, this implementation can be used instead, setting the **`_gas`** to some amount that may be changed and **`_maxCopy`** and **`_calldata`** to 0.

3S-OB-M03

Equality check should be performed for fees, not `>=` as it can lead to users sending more fees than supposed

Id	3S-OB-M03
Classification	Medium
Severity	High
Likelihood	Low
Category	Bug
Status	Addressed in #2f47c6c.

Description

In **`BRC20Factory::burn()`**, **`Vault::deposit()`**, it is required that the fee is equal to or exceeds the **`msg.value`**. However, this could lead to loss of fees, as the fee can be lowered without users noticing (or frontrunning them), and users would lose this part of the fee. In the case of the **`Vault`**, these funds would be left untracked because the fees are tracked in **`totalFees`**, so the owner would not be able to withdraw these extra funds.

Recommendation

Replace the `>=` checks for `==`. The equality check should be performed for fees, not `>=` as it can lead to users sending more fees than supposed

3S-OB-L01

Ownership in **`BRC20Factory`** could be transferred using a 2 step procedure, similarly to **`Vault`**

Id	3S-OB-L01
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Addressed in #ee04c91.

Description

It's recommended to change **`owner`** using a 2 step procedure to mitigate the risk of the contract losing ownership due to incorrect address.

Recommendation

Implement the 2 step ownership transfer procedure with a **`pendingOnwer`** variable and **`acceptOwnership()`** function. Here is an example implementation.

3S-OB-L02

Token addresses are not validated in **`BRC20Factory`** and **`Vault`**, so users can use incorrect tokens

Id	3S-OB-L02
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Status	Addressed in #1cd9410.

Description

`BRC20Factory::mint()`, **`BRC20Factory::burn()`**, **`Vault::deposit()`** and **`Vault::withdraw()`** allow sending any token, which could lead to problems if the token is not a **`BRC20`**.

Recommendation

Validate the sent token was created by the factory. To do this an **`isBRC20`** mapping should be added to **`BRC20Factory`** which stores all the tokens that were created by it.

3S-OB-L03

`BRC20Factory::burn()` could validate the receiver length

Id	3S-OB-L03
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Addressed in #d5f3226.

Description

`BRC20Factory::burn()` does not validate the `receiver`, which could lead to wasted funds. It's impossible to do a fail safe verification, but at least the length could be checked.

Recommendation

Revert if the length of the `receiver` is null.

3S-OB-L04

`BRC20Factory::addSigner()` and `Vault::addSigner()` must not allow adding the zero address as a signer

Id	3S-OB-L04
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Addressed in #4082cc6.

Description

The signature verification in `BRC20Factory` and `Vault` does not check that the recovered address is not null, which means that if `address(0)` is added as a signer, it would be possible to forge signatures.

Recommendation

In `BRC20Factory::addSigner()` and `Vault::addSigner()`, revert if an `address(0)` account is sent as argument.

3S-OB-L05

Fee event is missing in the constructors of **`BRC20Factory`** and **`Vault`**

Id	3S-OB-L05
Classification	Low
Severity	Low
Likelihood	High
Category	Suggestion
Status	Addressed in #edbf47e.

Description

The fee is set in the constructors of **`BRC20Factory`** and **`Vault`**, but the event is not emitted.

Recommendation

Emit the **`FeeChanged`** event in the constructor of both contracts.

3S-OB-L06

`BRC20Factory` constructor is missing a duplicate check

Id	3S-OB-L06
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Addressed in #ea46e07.

Description

`BRC20Factory` constructor does not check for duplicate signers, which would lead to problems as the index would be overriden.

Recommendation

Check for duplicate signers using the recommendation from #4.

3S-OB-L07

Domain separator calculation is not fork safe

Id	3S-OB-L07
Classification	Low
Severity	Medium
Likelihood	Low
Category	Bug
Status	Addressed in #143cee8.

Description

The domain separator is cached in the constructor of the contracts, which could lead to signature signing softwares in case of a hard fork. When a hard fork happens, **`block.chainid`** changes, which affects the domain separator, but as it is only computed in the constructor, it will be forever incorrect. Messages can still be signed but EIP712 will be broken.

Recommendation

The **`EIP712`** contract from OpenZeppelin handles this by caching the original domain separator and recomputing it if **`block.chainid`** has changed.

3S-OB-N01

Checks effects interactions pattern is now always followed

Id	3S-OB-N01
Classification	None
Category	Suggestion
Status	Addressed in #2f47c6c.

Description

State changes should always be performed before interacting with an external contract, or it increases the risk of reentrancy attacks.

Recommendation

In **`Vault::withdraw()`** decrease **`allowances[token]`** before sending the funds.

In **`Vault::withdrawFees()`** set **`totalFees`** to **`0`** before transferring the funds.

3S-OB-N02

`BRC20Factory` and **`Vault`** do different checks when removing signers

Id	3S-OB-N02
Classification	None
Category	Suggestion
Status	Acknowledged

Description

`BRC20Factory` checks if the **`account`** is authorized and if its index stored in **`indexes[account]`** is smaller than the length of **`signers`**. **`Vault`** requires that the length of signers is bigger than 1 and the **`account`** is authorized.

Recommendation

The check that the index of the account is smaller than the length of **`signers`** is not required and can be removed. Also, decide if it makes sense to check if the length of the signers is bigger than 1 in the **`BRC20Factory`** or if this check is not required in both contracts.

3S-OB-N03

Storage variables can be cached to save gas

Id	3S-OB-N03
Classification	None
Category	Optimization
Status	Addressed in #68ce26f.

Description

Reading/writing from the same storage variable more than once should be avoided to save gas.

Recommendation

`BRC20Factory::burn()` can cache the fee (read twice).

`Vault::deposit()` cache the fee (read thrice).

`Vault::acceptAdmin()` can cache the pendingAdmin (read 4 times).

`Vault::withdraw()` can cache signers.length (read number of signers times).

3S-OB-N04

`BRC20` parameters should be passed as arguments to the constructor of **`BRC20`** to save gas

Id	3S-OB-N04
Classification	None
Category	Optimization
Status	Addressed in #1fd3288.

Description

`BRC20Factory::createBRC20()` creates new **`BRC20`** tokens by setting the **`parameters`** storage variable to the **`name`**, **`symbol`** and **`decimals`** of the new token, and then the **`BRC20`** token in the constructor fetching this storage variable.

Recommendation

Send the **`name`**, **`symbol`** and **`decimals`** in the constructor to save gas, as **`create2`** will work just as fine, but it becomes significantly cheaper to deploy the tokens.

3S-OB-N05

Signer duplicate check can be performed by requiring signers being sent ordered

Id	3S-OB-N05
Classification	None
Category	Optimization
Status	Addressed in #b03ca97.

Description

Signers are checked for duplicates by creating a memory signers array and looping through this array for every new signer, which is inefficient.

Recommendation

Cache the last signer and assert that the current signer is bigger than the last, `require(signer > prevSigner, "duplicate")`. This is how Gnosis Safe does it, for example.

3S-OB-N06

`authorized` mapping is not required if the **`indexes`** mapping stores the indexes + 1

Id	3S-OB-N06
Classification	None
Category	Optimization
Status	Addressed in #dcd6efe.

Description

`BRC20Factory` and **`Vault`** store the **`signers`** array, the **`authorized`** and **`indexes`** mapping. The **`authorized`** mapping can be removed if the **`indexes`** mapping stores **`index + 1`** instead, so the check for existing signers can be made as **`indexes[signer] != 0`**.

Recommendation

Delete the **`authorized`** mapping, store the index as **`index + 1`** in the **`indexes`** mapping and check if the signer exists by doing **`indexes[signer] != 0`**.

3S-OB-N07

Reentrancy guard can be implemented with a **`uint256`** to save gas

Id	3S-OB-N07
Classification	None
Category	Optimization
Status	Addressed in #1cbaf62.

Description

Storing a **`uint256`** instead of a bool saves gas as it is cheaper changing storage from non null to non null value.

Recommendation

Implement the guard with a **`uint256`**. Here is an example implementation from OpenZeppelin.