ALEX Audit

Bitcoin Oracle and Bridge

October 2023

By CoinFabrik

v202307

Executive Summary

CoinFabrik was asked to audit the contracts for the ALEX project.

Bitcoin Oracle is a project with the goal of creating a tamper-proof and censorship-resistant indexing system for events within meta-protocols like BRC20. It relies on the Bitcoin chain as the ultimate source of truth, eliminating the necessity of depending on a single centralized off-chain indexer.

During this audit we found two high issues, and two minor issues. Also, two enhancements were proposed.

Three issues were resolved, and one mitigated. One of the enhancements was implemented.

Scope

The audited files are from the git repository located at the following repositories:

- https://github.com/alexgo-io/bitcoin-oracle at commit `4f8b451df3e0fd269b04eefab148ae428ae172ad`. Fixes reviewed on commit `88489987d706a3c0814017a2d9bf0ee0b2e4e231`.
- https://github.com/alexgo-io/bitcoin-bridge at commit `1cc34e39d082191d90e166b3199297007253c939`. Fixes reviewed on commit `daa01a8127965d289e7dc051d244033df12fdb7d`.

The scope for this audit includes and is limited to the following files:

- `bitcoin-oracle/packages/contracts/brc20-indexer-v1/contracts/indexer.clar`: The contract indexes BRC-20 transactions on the Bitcoin network.
- `bitcoin-oracle/packages/contracts/brc20-indexer-v1/contracts/indexer-registry.clar`: Storage contract for the indexer.
- `bitcoin-bridge/contracts/btc-bridge-endpoint.clar`: Bridge for exchanging BTC-aBTC.
- `bitcoin-bridge/contracts/btc-bridge-registry.clar`: Storage contract for the endpoint.

No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.

Methodology

CoinFabrik was provided with the source code, including automated tests that define the expected behavior, and general documentation about the project. Our auditors spent one week auditing the source code provided, which includes understanding the context of use, analyzing the boundaries of the expected behavior of each contract and function, understanding the implementation by the development team (including dependencies beyond the scope to be audited) and identifying possible situations in which the code allows the caller to reach a state that exposes some vulnerability. Without being limited to them, the audit process included the following analyses.

- Arithmetic errors
- Race conditions
- Reentrancy attacks
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

After delivering a report with our findings, the development team had the opportunity to comment on every finding and fix the issues they considered convenient. Once fixed and/or commented, our team ran a second review process to verify that the changes to the code effectively solve the issues found and do not unintentionally add new ones. This report includes the final status after the second review.

Findings

In the following table we summarize the security issues we found in this audit. The severity classification criteria and the status meaning are explained below. This table does not include the enhancements we suggest to implement, which are described in a specific section after the security issues.

ID	Title	Severity	Status
HI-01	Non-mined Transactions Processed	High	Resolved
HI-02	Peg-out not Payed Using Past Transaction	High	Resolved
MI-01	Authentication via tx-sender	Minor	Mitigated
MI-02	Hash Collision Because of Default Value	Minor	Resolved

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. Blocking bugs are also included in this category. They must be fixed **immediately**.
- **High:** These refer to a vulnerability that, if exploited, could have a substantial impact, but requires a more extensive setup or effort compared to critical issues. These pose a significant risk and demand immediate attention.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of, but might be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

An issue detected by this audit has one of the following statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code, but is a result of an intentional decision. The reported risk is accepted by the development team.
- **Resolved**: Adjusted program implementation to eliminate the risk.
- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk.

Critical Severity Issues

No issues found.

High Severity Issues

HI-01 Non-mined Transactions Processed

Location:

- `bitcoin-oracle/packages/contracts/brc20-indexer-v1/contracts/indexer.clar:182`

Classification:

- CWE-20: Improper Input Validation[1]

In the `index-tx-iter()` function, there is an issue related to the handling of the `verify-mined()` function's return values. The `verify-mined()` function can return one of three values: `(ok true)`, `(ok false)`, or an error. When it is true, the transaction was mined at the provided block, when false it was not, and an error is related to an error in the merkle proof. The code uses `try!` to check for errors but does not verify the boolean value returned wrapped in the response type. Therefore, the function does not fail if the transaction is not mined in the block.

If validators intentionally cooperate to sign a non-mined transaction, it can successfully pass through the indexer. This means that unmined and potentially malicious transactions can be processed without being properly validated, leading to potential security breaches and unauthorized actions within the system.

[1] https://cwe.mitre.org/data/definitions/20.html

Recommendation
Check the boolean value wrapped in the response type and fail if it is false.

Status
Resolved. Boolean checked.

HI-02 Peg-out not Payed Using Past Transaction

Location:
- `bitcoin-bridge/contracts/btc-bridge-endpoint.clar:242-280`

Classification:
- CWE-20: Improper Input Validation

The peg-out process for bridging BTC from Stacks to Bitcoin consists of the aBTC owner requesting it on the Stacks' smart contract and another actor claiming the request, transferring the BTC to the address of the requester and then finalizing the peg-out providing the transaction and necessary block data in order to verify the transaction was mined.
However, when finalizing the peg-out, the contract does not check if the transaction provided is actually related to the peg-out. Since there is no association between the address on Stacks and the address on Bitcoin, the only validation on the transaction is that the requester received the amount he asked for and the Bitcoin address provided when claiming is in the outputs. Therefore, an attacker can use any transaction which has not been used for a peg-out before, while the output for the requester has the same value as the requested, without actually transferring the BTC.

Steps to Replicate
- Monitor the contract for a new peg-out request.
- Search into the requester's transaction history on Bitcoin for a transaction where the output for that user has the same value as the requested.
- Claim the peg-out providing the attacker's Stacks address and original Bitcoin address from the transaction found in requester's history.
- Finalize the peg-out providing the transaction and block data from the transaction found in requester's history.

Recommendation
This issue can be mitigated by storing the burn block height when the peg-out is claimed and then validating the block headers provided when finalizing the peg-out against the stored height.

Status
Resolved. Fixed according to the recommendation.

Medium Severity Issues

No issues found.

Minor Severity Issues

MI-01 Authentication via tx-sender

Location:

- `bitcoin-oracle/packages/contracts/brc20-indexer-v1/contracts/indexer.clar`
- `bitcoin-oracle/packages/contracts/brc20-indexer-v1/contracts/indexer-registry.clar`
- `bitcoin-bridge/contracts/btc-bridge-endpoint.clar`
- `bitcoin-bridge/contracts/btc-bridge-registry.clar`

The system utilizes `tx-sender` for its authentication processes. This method, while functional, presents latent vulnerabilities, particularly exposing actors within the system to threats known as phishing[2].

Actors could inadvertently activate a malicious contract. Once activated, the deceptive contract can access and initiate certain functions, presenting actions as if they were done by the original actor. This impersonation potential poses risks, depending on the specific function being accessed.

The system only uses them for owner authentication. Therefore, it is less likely to be exploited while using a dedicated account as owner.

Recommendation

It is advisable to switch from using `tx-sender` to `contract-caller` for a more reliable and secure authentication method. Furthermore, introducing a mapping for trusted callers can add an extra layer of security, particularly if the system needs to interact with specific intermediary contracts.

Status

Mitigated. The contract owner will be the Decentralized Autonomous Organization (DAO) deployed by ALEX. Therefore, the contract calls will be voted by the participants.

MI-02 Hash Collision Because of Default Value

Location:

[2] https://www.coinfabrik.com/blog/tx-sender-in-clarity-smart-contracts/

- `bitcoin-oracle/packages/contracts/brc20-indexer-v1/contracts/indexer.clar`

The current implementation of the `hash-tx()` function in the codebase uses a `default-to` mechanism to handle cases where a value is too large to fit within the specified buffer. While this approach may seem convenient, it introduces a potential security risk in the form of hash collisions.

A hash collision occurs when two different inputs produce the same hash output. In this case, if the value assigned to the argument exceeds the buffer's capacity, it is set to a default value (`0x`), potentially causing multiple different inputs to produce the same hash output.

Recommendation
Since the types defined cannot exceed the buffer's capacity, it is better to use `unwrap-panic` for unwrapping the hash value.

Status
Resolved. `unwrap-panic` implemented instead of default value.

Enhancements

These items do not represent a security risk. They are best practices that we suggest implementing.

ID	Title	Status
EN-01	Unused Pausing Mechanism	Implemented
EN-02	Unused Utility Functions	Not implemented

EN-01 Unused Pausing Mechanism

Location:

- `bitcoin-oracle/packages/contracts/brc20-indexer-v1/contracts/indexer-registry.clar`

The indexer registry defines pausing-related variables, errors and functions, but they are not used in the operative functions. Therefore, pausing the contract does not stop any functionality.

Recommendation
Either remove the pausing mechanism or implement it for stopping the operative functions when setting to `true`.

Status
Implemented. Pausing mechanism implemented.

EN-02 Unused Utility Functions

Location:

- `bitcoin-bridge/contracts/btc-bridge-endpoint.clar`

The indexer registry defines mathematical utility functions `min` and `div-down`, but these functions are not used by any of the public functions.

Recommendation
Remove `min` and `div-down` functions.

Status
Not implemented.

Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest to other stakeholders of the project, including users of the audited contracts, token holders or project investors.

Centralization

The two systems are not completely autonomous. They require backends for validating transactions in the bitcoin oracle, and another for sending Bitcoin transactions in order to finalize peg-outs.

Upgrades

Since the implementations are separated from the storage and can be paused, the two systems can be upgraded.

Changelog

- 2023-10-30 – Initial report based on commits `4f8b451df3e0fd269b04eefab148ae428ae172ad` and `1cc34e39d082191d90e166b3199297007253c939`.
- 2023-11-06 – Final report based on commits `88489987d706a3c0814017a2d9bf0ee0b2e4e231` and `daa01a8127965d289e7dc051d244033df12fdb7d`.

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the ALEX project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.