



Defense By Thesis

Security Audit Report

Best-in-Slot

brc20-programmable-
module

Initial Report // August 29, 2025

Final Report // September 12, 2025






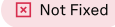





















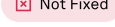


Team Members

PR // Senior Security Auditor

Solthodox // Security Auditor



Table of Contents

<u>1.0 Scope</u>	3
<u>1.1 Technical Scope</u>	
<u>2.0 Executive Summary</u>	4
<u>2.1 Schedule</u>	
<u>2.2 Overview</u>	
<u>2.3 Threat Model</u>	
<u>2.4 Tests & Project Documentation</u>	
<u>3.0 Key Findings Table</u>	6
<u>4.0 Findings</u>	7
<u>4.1 BlockHistoryCache::latest() Does Not Distinguish Between Empty Cache and None Value</u>	
 Low  Fixed	
<u>4.2 Brc20ProgDatabase::reorg() Reaches Reorg Depths Exceeding MAX_REORG_HISTORY_SIZE</u>	
 Low  Fixed	
<u>4.3 Non-functional Default Implementation for Brc20ProgDatabase</u>	
 None  Not Fixed	
<u>4.4 Inconsistency Between Code and Comments</u>	
 None  Fixed	
<u>4.5 Non-idiomatic Conversion of enum to String</u>	
 None  Fixed	
<u>4.6 Long Argument Lists for Functions</u>	
 None  Fixed	
<u>4.7 BytecodeED Decode Can Panic</u>	
 None  Fixed	
<u>4.8 Redundant Implementation of Functionality in Libraries</u>	
 None  Fixed	
<u>4.9 Brc20ProgDatabase Permits Construction of Invalid States</u>	
 None  Fixed	
<u>4.10 RPC Uses Optional HTTP Basic Authentication</u>	
 None  Fixed	
<u>4.11 Silent Error Swallowing</u>	
 None  Fixed	
<u>4.12 Lack of RPC Response Validation</u>	
 None  Fixed	
<u>4.13 Ticker Encoding Inconsistency</u>	
 None  Fixed	
<u>4.14 Limited BTC Wallet Address Support</u>	
 None  Not Fixed	
<u>4.15 No Signature Size Validation</u>	
 None  Fixed	
<u>5.0 Appendix A</u>	16
<u>5.1 Severity Rating Definitions</u>	
<u>6.0 Appendix B</u>	18
<u>6.1 Thesis Defense Disclaimer</u>	

About Thesis Defense

Defense is the security auditing arm of Thesis, Inc., the venture studio behind tBTC, Fold, Mezo, Acre, Tahoe, Etcher, and Embody. At [Defense](#), we fight for the integrity and empowerment of the individual by strengthening the security of emerging technologies to promote a decentralized future and user freedom. Defense is the leading Bitcoin applied cryptography and security auditing firm. Our [team](#) of security auditors have carried out hundreds of security audits for decentralized systems across a number of ecosystems including Bitcoin, Ethereum + EVMs, Stacks, Cosmos SDK, NEAR and more. We offer our services within a variety of technologies including smart contracts, bridges, cryptography, node implementations, wallets and browser extensions, and dApps.

Defense will employ the [Defense Audit Approach](#) and [Audit Process](#) to the in scope service. In the event that certain processes and methodologies are not applicable to the in scope services, we will indicate as such in individual audit or design review SOWs. In addition, Thesis Defense provides clear guidance on successful [Security Audit Preparation](#).

Section 1.0 Scope

Technical Scope

- **Repository:** <https://github.com/bestinslot-xyz/brc20-programmable-module>
- **Audit Commit:** `4ab801b940a721c5ffd0126c70bcb9c4821fa1f8`
- **Verification Commit:** `e6e81baaa3a25b067e46aaf44856a6da57d460b4`
- **Folders/files in-Scope:**
 - `src/brc20_controller`
 - `src/db`
 - `src/engine/precompiles`
 - `src/server/rpc_server.rs`



Section 2.0

Executive Summary

Schedule

This security audit was conducted from August 15, 2025 to August 29, 2025 by 2 security auditors for a total of 16 person-days.

Overview

The BRC20 programmable module is a Rust-based smart contract execution engine that brings programmability to the BRC20 standard on Bitcoin. It allows users to inscribe smart contracts and function calls on the Bitcoin blockchain using a custom EVM execution engine powered by revm.

The module supports both Ethereum-compatible methods and custom methods for BRC20 indexer integration. It enables users to deploy contracts via inscriptions, make contract calls, and deposit/withdraw BRC20 tokens through Bitcoin inscriptions sent to `OP_RETURN "BRC20PROG"`.

db

db contains database management components that handle persistent storage of blockchain state. It supports operations like `brc20_commitToDatabase` to write pending changes to disk and `brc20_clearCaches` to remove pending changes and revert to the last saved state.

Server

The server provides Ethereum-compatible `eth_*` methods for querying blocks and transactions, plus custom `brc20_*` methods for BRC20 indexer integration.

BRC-20 Controller

The `BRC20_Controller` handles BRC20 token deposits, transfers, and withdrawals, serving as the bridge between traditional BRC20 tokens and the programmable execution environment.

EVM Precompiles

The core EVM execution engine is using `revm` for deterministic smart contract execution. The precompiled contracts enable Bitcoin-specific functionality. The precompiles allow smart contracts to interact with Bitcoin blockchain data and maintain awareness of transactions happening outside the execution engine.

Threat Model

We analyzed and tested the following areas for security vulnerabilities:

Chain Synchronization & Consensus Risks

- Long reorgs / deep chain attacks: Handling of reorganizations may fail under large, unlikely but possible rollbacks.
- Network partitioning: If the system sees only part of the network, it may accept invalid states.

Smart Contract Logic Risks

- Incorrect assumptions about liveness/finality: Contracts may assume a transaction is final too early, exposing them to reorg risks.
- Boundary/overflow errors: Arithmetic or gas accounting mistakes in pre-compiled contracts could be exploited.



- Unintended DoS vectors: Certain inputs could cause excessive computation, storage bloat, or transaction rejection.

Database & State Risks

- Corruption under concurrent access: Multiple processes modifying state during a reorg could cause inconsistencies.
- Rollback mismatches: If the database rolls back differently than the chain, data divergence may occur.
- Persistence guarantees: Crashes during reorg or serialization may leave the database in an unrecoverable state.

Serialization / Deserialization Risks

- Input fuzzing attacks: Crafted inputs could trigger crashes, infinite loops, or memory exhaustion.
- Ambiguities in encoding: Malicious data may be deserialized differently by different components, leading to replay or consensus failures.
- Backward compatibility: Future upgrades to serialization formats may break assumptions.

Operational & Integration Risks

- Dependency trust: Reliance on external libraries for parsing, database management, or Bitcoin integration could introduce vulnerabilities.
- Logging and monitoring gaps: Limited visibility into reorg events or serialization errors may delay detection of attacks.
- Key management: If contracts interact with signing infrastructure, weak key handling could be exploited.

Tests & Project Documentation

While basic functionality tests covering standard use cases are implemented, the test suite lacks comprehensive coverage for edge cases and error conditions, leaving potential vulnerabilities and failure modes unvalidated. We found that the project documentation is sufficient, however, it is not always accurate. We recommend that the project documentation always reflect the current state of the code base.

Section 3.0

Key Findings Table

Issues	Severity	Status
ISSUE #1 <code>BlockHistoryCache::latest()</code> Does Not Distinguish Between Empty Cache and <code>None</code> Value	✓ Low	✓ Fixed
ISSUE #2 <code>Brc20ProgDatabase::reorg()</code> Reaches Reorg Depths Exceeding <code>MAX_REORG_HISTORY_SIZE</code>	✓ Low	✓ Fixed
ISSUE #3 Non-functional Default Implementation for <code>Brc20ProgDatabase</code>	⇓ None	✗ Not Fixed
ISSUE #4 Inconsistency Between Code and Comments	⇓ None	✓ Fixed
ISSUE #5 Non-idiomatic Conversion of <code>enum</code> to <code>String</code>	⇓ None	✓ Fixed
ISSUE #6 Long Argument Lists for Functions	⇓ None	✓ Fixed
ISSUE #7 <code>BytecodeED Decode</code> Can Panic	⇓ None	✓ Fixed
ISSUE #8 Redundant Implementation of Functionality in Libraries	⇓ None	✓ Fixed
ISSUE #9 <code>Brc20ProgDatabase</code> Permits Construction of Invalid States	⇓ None	✓ Fixed
ISSUE #10 RPC Uses Optional HTTP Basic Authentication	⇓ None	✓ Fixed
ISSUE #11 Silent Error Swallowing	⇓ None	✓ Fixed
ISSUE #12 Lack of RPC Response Validation	⇓ None	✓ Fixed
ISSUE #13 Ticker Encoding Inconsistency	⇓ None	✓ Fixed
ISSUE #14 Limited BTC Wallet Address Support	⇓ None	✗ Not Fixed
ISSUE #15 No Signature Size Validation	⇓ None	✓ Fixed

Severity definitions can be found in [Appendix A](#)



Section 4.0

Findings

We describe the security issues identified during the security audit, along with their potential impact. We also note areas for improvement and optimizations in accordance with best practices. This includes recommendations to mitigate or remediate the issues we identify, in addition to their status before and after the fix verification.

ISSUE#1

BlockHistoryCache::latest() Does Not Distinguish Between Empty Cache and None Value

✓ Low

✓ Fixed

Location

[src/db/cached_database/block_history_cache.rs#L67-L69](#)

Description

The cache uses `BTreeMap<u64, Option<V>>`.

Calling `self.cache.values().last().cloned()` returns an `Option<Option<V>>`. Using `.unwrap_or(None)` on this value returns an `Option<V>` where both `Some(None)` and `None` result in `None` being returned. This means that when `cache.latest()` is called, there is no distinction between the cache being empty and the value in the cache being `None`.

Impact

If the cache is emptied of values but not deleted this can result in the database contents being overwritten with a `None` value in `BlockCachedDatabase::commit()`. Other functions calling `cache.latest()` can return incorrect results. `BlockHistoryCache::unset()` will fail to insert a `None` value to an empty cache.

Recommendation

In `BlockHistoryCache::latest()`, we recommend distinguishing between no value being recorded in the cache, and the cache returning a `None` value. Using an alternative datatype such as `Either<(), V>` is recommended to help distinguish the cases. We also recommend adding tests confirming correct behavior in this situation.



ISSUE#2

Brc20ProgDatabase::reorg() Reaches Reorg Depths Exceeding MAX_REORG_HISTORY_SIZE

✓ Low

✓ Fixed

Location

[src/db/brc20_prog_database.rs#L1023](#)

Description

After `clear_caches()` is called, the latest block number is set to `None`. When this happens in a reorg, the latest block height is determined from `self.db_block_number_to_hash` which has had its recent values cleared in the reorg. Thus, a subsequent reorg may exceed the maximum depth.

Impact

The cache implementation assumes that reorgs cannot exceed the `MAX_REORG_HISTORY_SIZE`. Exceeding the size will invalidate guarantees made based on that assumption.

Recommendation

When performing a reorg, keep a record of the previous highest block, and compare any subsequent reorg attempts to the previously highest block rather than the current latest block.



ISSUE#3

Non-functional Default Implementation for Brc20ProgDatabase

None

Not Fixed

Location

src/db/brc20_prog_database.rs

Description

`Brc20ProgDatabase::default()` creates an uninitialized database that is not functional. This results in each database in `Brc20ProgDatabase` being wrapped in an `Option`, resulting in checks for a `None` value whenever any of the databases is used.

Impact

None - Informational.

Recommendation

We recommend using RAII (resource acquisition is initialization) for `Brc20ProgDatabase` and remove the `Option<>` wrapping of individual databases. This improves readability and eliminates redundant runtime checks.

Verification Status

The Best-in-Slot team stated that this is intended behavior that is necessary for using `mem::take(...)` when passing to the EVM. The revm engine requires ownership and `mem::take` requires a Default implementation. The Default implementation cannot construct a functional database from scratch, because there can only be one primary reader. Currently, if a caller tries to use the non-functional implementation, it would panic.



ISSUE#4

Inconsistency Between Code and Comments

None

Fixed

Location

[src/db/cached_database/block_cached_database.rs#L181-L238](#)

[src/db/types/uint_ed.rs#L25](#)

[src/db/cached_database/block_history_cache.rs#L34-L38](#)

Description

- BlockCachedDatabase::commit() and BlockCachedDatabase::reorg() call BlockCachedDatabase::clear_cache() contrary to the comments
- uint_ed.rs#L25: U512ED has 8 limbs
- block_history_cache.rs#L34-L38The applied filter is “less than or equal to”. The last key returned from the filter is ignored when removing keys regardless of its age.

Recommendation

We recommend updating the comments to reflect the current functionality of the code.

ISSUE#5

Non-idiomatic Conversion of enum to String

None

Fixed

Location

[src/engine/utills.rs#L153-L229](#) [src/db/types/tx_receipt_ed.rs#L17-L21](#)

Description

The enum `ExecutionResult`, `SuccessReason` and `HaltReason` are converted to `String` and stored in memory as `String`. This can affect performance and removes the type soundness of using an enum.

Impact

None - Informational.

Recommendation

We recommend using the applicable enums to avoid storing unnecessary strings in memory. The values can also optionally be encoded as constant-size values of smaller size in the Encode/Decode implementations.



ISSUE#6

Long Argument Lists for Functions

None

Fixed

Location

[src/db/types/tx_ed.rs#L79-L93](#)

[src/db/types/tx_receipt_ed.rs#L71-L89](#)

[src/db/types/block_ed.rs#L149-L165](#)

Description

Large numbers of arguments for a function call reduce legibility and increase the risk of errors, especially when multiple arguments of the same type are present.

Impact

None - Informational.

Recommendation

We recommend grouping related arguments, such as `v`, `r`, and `s` values, into structs to reduce the number of arguments.

ISSUE#7

BytecodeED Decode Can Panic

None

Fixed

Location

[src/db/types/bytecode_ed.rs#L29](#) [src/db/types/bytecode_ed.rs#L58](#)

Description

`Bytecode::new_raw()` will panic if the input is in the incorrect format.

`Bytecode::new_raw_checked()` returns an error if the input is in incorrect format.

Recommendation

We recommend determining if panicking is the desired behavior and using `new_raw_checked()` if an error is desired.



ISSUE#8

Redundant Implementation of Functionality in Libraries

None

Fixed

Location

[src/db/cached_database/block_history_cache.rs#L115-L123](#)

[src/db/types/uint_ed.rs](#)

Description

- Identical, redundant code to `BTreeMap::retain()`.
- `alloy_primitives::aliases` providing matching types and encoding/decoding functions.

Impact

None - Informational.

Recommendation

We recommend removing duplicate code.

ISSUE#9

Brc20ProgDatabase Permits Construction of Invalid States

None

Fixed

Location

[src/db/brc20_prog_database.rs](#)

[src/db/brc20_prog_database.rs#L685-L696](#)

[src/engine/engine.rs#L628-L631](#)

Description

`Brc20ProgDatabase` expects block hash, mine timestamp, gas used, and block timestamp to stay consistent. Having separate functions for them, instead of a single function taking all four values as arguments, permits the construction of an invalid state.

Impact

None - Informational.

Recommendation

We recommend replacing the functions `set_block_hash`, `set_mine_timestamp`, `set_gas_used`, and `set_block_timestamp` with a single function that sets all values in one call, removing the possibility of erroneous state construction.



ISSUE#10

RPC Uses Optional HTTP Basic Authentication

None

Fixed

Location

[src/server/auth.rs](#)

[src/server/start.rs](#)

Description

HTTP Basic Authentication is unsuitable for environments where transmission of credentials in plaintext headers is unacceptable. Authentication is optional. Using more secure authentication is rejected for performance reasons.

Impact

None - Informational.

Recommendation

We recommend documenting the limits on appropriate usage of the RPC API.

ISSUE#11

Silent Error Swallowing

None

Fixed

Location

[src/server/rpc_server#294](#)

Description

The implementation uses a double `unwrap_or` pattern that obscures error handling logic and masks the underlying error from `get_contract_address_by_inscription_id`. This pattern makes debugging difficult by silently converting errors into default values without providing visibility into what went wrong.

Impact

None - Informational.

Recommendation

We recommend replacing the double `unwrap_or` pattern with explicit error propagation. We recommend using error handling that logs the original error before applying fallback behavior, ensuring developers have visibility into failure modes during debugging and monitoring.



ISSUE#12

Lack of RPC Response Validation

None

Fixed

Location

[src/engine/precompiles/brc20_balance_precompile.rs#51](#)

Description

The BRC20 balance precompile lacks input validation when processing server responses, directly parsing the returned data without verifying it contains a valid numeric string. This oversight could lead to parsing failures if the server returns unexpected content such as HTML error pages, JSON error objects, or malformed data.

Impact

None - Informational.

Recommendation

We recommend adding checks to ensure the response contains only valid numeric data and handle common error scenarios such as HTML error pages, empty responses, or non-numeric strings.

ISSUE#13

Ticker Encoding Inconsistency

None

Fixed

Location

[src/engine/precompiles/brc20_balance_precompile.rs#46](#)

Description

There is a potential encoding inconsistency in the ticker parameter handling within the `get_brc20_balance` function. The function receives ticker as `& Bytes` from the ABI-decoded smart contract call, which is already in binary format, but then applies `hex::encode(ticker)` when constructing the HTTP query. This creates ambiguity about the expected input format from smart contracts.

Impact

None - Informational.

Recommendation

We recommend clarifying the precompile interface specification to explicitly state that the first argument should be a `bytes32` hex-encoded ticker in order to prevent inconsistent encoding.



ISSUE#14

Limited BTC Wallet Address Support

None

Not Fixed

Location

[src/engine/precompiles/brc20_verify_precompile.rs](#)

Description

Legacy P2PKH addresses (starting with “1”) are not supported, nor are multisig scripts. Furthermore, a contract failure for this reason will be interpreted as invalid signature.

Impact

None - Informational.

Recommendation

We recommend expanding address format support to include legacy P2PKH addresses and common multisig scripts where feasible. Additionally, we recommend implementing explicit error handling that distinguishes between invalid signatures and unsupported address formats, providing clear feedback to users and developers when address types are not supported by the verification system.

Verification Status

The Best-in-Slot team considers this low priority and will wait for BIP 322 library support rather than rewrite it.

ISSUE#15

No Signature Size Validation

None

Fixed

Location

[src/engine/precompiles/brc20_verify_precompile.rs#44](#)

Description

The signature verification precompile processes and slices signature data without validating the input size, creating potential denial-of-service attack vectors. Malicious actors could submit excessively large signatures that consume significant memory and processing resources during verification operations, potentially impacting system performance or causing resource exhaustion.

Impact

None - Informational.

Recommendation

We recommend adding a signature size limit validation.





Section 5.0

Appendix A

Severity Rating Definitions

At Thesis Defense, we utilize the [ImmuneFi Vulnerability Severity Classification System - v2.3](#).

Severity	Definition
 Critical	<ul style="list-style-type: none">• Execute arbitrary system commands• Retrieve sensitive data/files from a running server, such as:<ul style="list-style-type: none">◦ /etc/shadow◦ database passwords◦ blockchain keys (this does not include non-sensitive environment variables, open source code, or usernames)• Taking down the application/website• Taking down the NFT URI• Taking state-modifying authenticated actions (with or without blockchain state interaction) on behalf of other users without any interaction by that user, such as:<ul style="list-style-type: none">• Changing registration information• Commenting• Voting• Making trades• Withdrawals, etc.• Changing the NFT metadata• Subdomain takeover with already-connected wallet interaction• Direct theft of user funds Malicious interactions with an already-connected wallet, such as:<ul style="list-style-type: none">• Modifying transaction arguments or parameters• Substituting contract addresses• Submitting malicious transactions• Direct theft of user NFTs• Injection of malicious HTML or XSS through NFT metadata
 High	<ul style="list-style-type: none">• Injecting/modifying the static content on the target application without JavaScript (persistent), such as:<ul style="list-style-type: none">◦ HTML injection without JavaScript◦ Replacing existing text with arbitrary text◦ Arbitrary file uploads, etc.• Changing sensitive details of other users (including modifying browser local storage) without already-connected wallet interaction and with up to one click of user interaction, such as:<ul style="list-style-type: none">◦ Email or password of the victim, etc.◦ Improperly disclosing confidential user information, such as:<ul style="list-style-type: none">▪ Email address▪ Phone number▪ Physical address, etc.<ul style="list-style-type: none">▪ PSubdomain takeover without already-connected wallet interaction



Severity	Definition
<div> <div></div> <div>Medium</div> </div>	<ul style="list-style-type: none"> Changing non-sensitive details of other users (including modifying browser local storage) without already-connected wallet interaction and with up to one click of user interaction, such as: <ul style="list-style-type: none"> Changing the first/last name of user Enabling/disabling notifications Injecting/modifying the static content on the target application without JavaScript (reflected), such as: <ul style="list-style-type: none"> Reflected HTML injection Loading external site data Redirecting users to malicious websites (open redirect)
<div> <div></div> <div>Low</div> </div>	<ul style="list-style-type: none"> Changing details of other users (including modifying browser local storage) without already-connected wallet interaction and with significant user interaction, such as: <ul style="list-style-type: none"> Iframing leading to modifying the backend/browser state (must demonstrate impact with PoC) Taking over broken or expired outgoing links, such as: <ul style="list-style-type: none"> Social media handles, etc Temporarily disabling user to access target site, such as: <ul style="list-style-type: none"> Locking up the victim from login Cookie bombing, etc.
<div> <div></div> <div>None</div> </div>	<ul style="list-style-type: none"> We make note of issues of no severity that reflect best practice recommendations or opportunities for optimization, including, but not limited to, gas optimization, the divergence from standard coding practices, code readability issues, the incorrect use of dependencies, insufficient test coverage, or the absence of documentation or code comments.



Section 6.0

Appendix B

Thesis Defense Disclaimer

Thesis Defense conducts its security audits and other services provided based on agreed-upon and specific scopes of work (SOWs) with our Customers. The analysis provided in our reports is based solely on the information available and the state of the systems at the time of review. While Thesis Defense strives to provide thorough and accurate analysis, our reports do not constitute a guarantee of the project's security and should not be interpreted as assurances of error-free or risk-free project operations. It is imperative to acknowledge that all technological evaluations are inherently subject to risks and uncertainties due to the emergent nature of cryptographic technologies.

Our reports are not intended to be utilized as financial, investment, legal, tax, or regulatory advice, nor should they be perceived as an endorsement of any particular technology or project. No third party should rely on these reports for the purpose of making investment decisions or consider them as a guarantee of project security.

Links to external websites and references to third-party information within our reports are provided solely for the user's convenience. Thesis Defense does not control, endorse, or assume responsibility for the content or privacy practices of any linked external sites. Users should exercise caution and independently verify any information obtained from third-party sources.

The contents of our reports, including methodologies, data analysis, and conclusions, are the proprietary intellectual property of Thesis Defense and are provided exclusively for the specified use of our Customers. Unauthorized disclosure, reproduction, or distribution of this material is strictly prohibited unless explicitly authorized by Thesis Defense. Thesis Defense does not assume any obligation to update the information contained within our reports post-publication, nor do we owe a duty to any third party by virtue of making these analyses available.

