MiloTruck

SovaBTC

Security Review

June 23, 2025

Contents

1 Introduction

1.1 About MiloTruck

MiloTruck is an independent security researcher, primarily working as a Lead Security Researcher at Spearbit and Cantina. Previously, he was part of the team at Renascence Labs and a Lead Auditor at Trust Security.

For private audits or security consulting, please reach out to him on Twitter *@milotruck*.

1.2 Disclaimer

A smart contract security review **can never prove the complete absence of vulnerabilities**. Security reviews are a time, resource and expertise bound effort to find as many vulnerabilities as possible. However, they cannot guarantee the absolute security of the protocol in any way.

2 Risk Classification

Severity Level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2.1 Impact

- High - Funds are **directly** at risk, or a **severe** disruption of the protocol's core functionality.
- Medium - Funds are **indirectly** at risk, or **some** disruption of the protocol's functionality/availability.
- Low - Funds are **not** at risk.

2.2 Likelihood

- High - Highly likely to occur.
- Medium - Might occur under specific conditions.
- Low - Unlikely to occur.

3 Executive Summary

3.1 About SovaBTC

Sova is an EVM blockchain with integrated Bitcoin interoperability designed for native BTC wealth products. Sova's architecture allows smart contracts to use BTC UTXOs programmatically, enabling a new wave of BTC-based DeFi applications.

Key Features:

- **Native Bitcoin Integration**: Direct UTXO interaction using custom precompiles
- **EVM Compatibility**: Full support for Ethereum-style smart contracts
- **Permissionless**: Open protocol for building Bitcoin-native applications
- **Institutional Grade**: Built for serious Bitcoin wealth products

3.2 Overview

Project Name	SovaBTC
Project Type	Bitcoin L2
Language	Solidity, Rust
Repository	SovaNetwork/contracts
Commit Hash	d0cf19abc7b115ea195a93136d01f35fc84c3bad

3.3 Issues Found

High	1
Medium	2
Low	2
Informational	2

4 Findings

4.1 High Risk

4.1.1 `UBTC.depositBTC()` can be called with the same `signedTx` multiple times

Context: UBTC.sol#L121-L144

Description: In the `UBTC` contract, users call `depositBTC()` with a `signedTx` to receive `UBTC` in exchange for BTC deposited on the bitcoin network.

However, the function does not validate that `signedTx` has not been used for a prior deposit. This means that `depositBTC()` can be called with the same `signedTx` multiple times, allowing users to mint infinite UBTC using one deposit transaction.

Recommendation: Include replay protection in `depositBTC()`. For example, the function could check that the transaction hash/id of the provided `signedTx` has not been used before.

SovaBTC: Fixed in PR 11.

MiloTruck: Verified, a `usedTxids` mapping has been added and enforced in `depositBTC()` for replay protection.

4.2 Medium Risk

4.2.1 Mismatching types between sova-reth and solidity contracts

Context:

- SovaBitcoin.sol#L32-L37
- abi.rs#L102
- UBTC.sol#L160
- UBTC.sol#L187-L188
- abi.rs#L33

Description: In the `SovaBitcoin` library, `outputIndex` is declared as a `uint32` in the `Input` struct:

```
struct Input {
    bytes32 prevTxHash;
    uint32 outputIndex;
    bytes scriptSig;
    bytes[] witness;
}
```

However, in the sova-reth client, `output_index` is declared as `uint256`:

```
struct Input {
    bytes32 prev_tx_hash;
    uint256 output_index;
    bytes script_sig;
    bytes[] witness;
}
```

As such, if the BTC precompile is ever called to decode a transaction where an input's `outputIndex` is greater than `uint32.max`, the call would not revert when it should (as `outputIndex` should never be larger than 4 bytes).

In `UBTC.withdraw()`, `btcBlockHeight` is encoded as a `uint32` when calling the BTC precompile:

```
function withdraw(uint64 amount, uint64 btcGasLimit, uint32 btcBlockHeight, string calldata dest)
```

```
bytes memory inputData =
    abi.encode(SovaBitcoin.UBTC_SIGN_TX_BYTES, msg.sender, amount, btcGasLimit, btcBlockHeight, dest);
```

However, in the sova-reth client, `block_height` is decoded as a `uint64`:

```
let input_type = DynSolType::Tuple(vec![
    DynSolType::FixedBytes(4), // method selector
    DynSolType::Address,       // caller address
    DynSolType::Uint(64),      // amount
    DynSolType::Uint(64),      // btcGasLimit
    DynSolType::Uint(64),      // block_height
    DynSolType::String,        // destination
]);
```

Fortunately, this does not have any impact as `btcBlockHeight` is encoded as 32 bytes in `UBTC.withdraw()`.

Recommendation: In the sova-reth client, since an input's `output_index` should never exceed 4 bytes, declare `output_index` as a `uint32`:

```
  struct Input {
      bytes32 prev_tx_hash;
-     uint256 output_index;
+     uint32 output_index;
      bytes script_sig;
      bytes[] witness;
  }
```

In `UBTC.withdraw()`, declare `btcBlockHeight` as a `uint64`:

```
- function withdraw(uint64 amount, uint64 btcGasLimit, uint32 btcBlockHeight, string calldata dest)
+ function withdraw(uint64 amount, uint64 btcGasLimit, uint64 btcBlockHeight, string calldata dest)
```

SovaBTC: Fixed in PR 12 and PR 84.

MiloTruck: Verified, the recommendation has been implemented.

4.2.2 Missing update function for `maxGasLimitAmount` in `UBTC`

Context:

- UBTC.sol#L29-L30
- UBTC.sol#L79

Description: In the `UBTC` contract, `maxGasLimitAmount` is initialized to 50,000,000 sats in the constructor:

```
maxGasLimitAmount = 50_000_000; // 0.5 BTC (50,000,000 sats)
```

However, there is no function to update the value of `maxGasLimitAmount` after deployment. As such, it will always remain at a fixed value 50,000,000 sats and cannot be updated by the admin.

Recommendation: Add the missing `setMaxGasLimitAmount()` function to the contract.

SovaBTC: Fixed in PR 8 and PR 12.

MiloTruck: Verified, a new `setMaxGasLimitAmount()` function was added to the contract.

4.3 Low Risk

4.3.1 `SovaBitcoin.isValidDeposit()` wrongly assumes `lockTime` is always a timestamp

Context:

- SovaBitcoin.sol#L135-L137
- script.h#L45-L47

Description: `SovaBitcoin.isValidDeposit()` checks that a bitcoin transaction's `lockTime` is not greater than the current `block.timestamp`:

```
if (btcTx.locktime > block.timestamp) {
    revert InvalidLocktime();
}
```

This is meant to ensure that transactions which can only be included in future blocks are not valid.

However, `lockTime` is not always a timestamp; it can also be a block number, as seen in Bitcoin Core:

```
// Threshold for nLockTime: below this value it is interpreted as block number,
// otherwise as UNIX timestamp.
static const unsigned int LOCKTIME_THRESHOLD = 500000000; // Tue Nov  5 00:53:20 1985 UTC
```

More specifically:

- If `lockTime < 500000000`, `lockTime` is a block number.
- If `lockTime >= 500000000`, `lockTime` is a timestamp.

As such, if `isValidDeposit()` is called with a transaction where its `lockTime` is a future block number (i.e. smaller than `500000000`, but greater than the current bitcoin block height), the check shown above would wrongly pass.

Recommendation: Modify the check to have the following logic:

- If `lockTime < 500000000`, it should not be greater than `SovaL1Block.currentBlockHeight`.
- Otherwise, `lockTime <= block.timestamp` should be true.

SovaBTC: Fixed in PR 14.

MiloTruck: Verified, `isValidDeposit()` now ensures that all transactions have a `lockTime` of `0`.

4.3.2 Security assumptions regarding the BTC precompile and sova-reth client

Context:

- SovaBitcoin.sol#L78-L80
- SovaBitcoin.sol#L101-L104
- UBTC.sol#L187-L193

Description: The following are assumptions that must hold true in order for the `UBTC` contract to be safe.

(1) A call to the BTC precompile's `CheckSignature` function (i.e. selector `0x3`) must revert if the provided `signedTx` has an invalid signature:

```
function checkSignature(bytes calldata signedTx) internal view returns (bool success) {
    (success,) = BTC_PRECOMPILE.staticcall(abi.encodePacked(CHECKSIG_BYTES, signedTx));
}
```

Otherwise, `SovaBitcoin.checkSignature()` could incorrectly return `true` for a invalid `signedTx`.

(2) A call to the BTC precompile's `BroadcastTransaction` function (i.e. selector `0x1`) must revert if broadcasting the transaction fails:

```
function broadcastBitcoinTx(bytes memory signedTx) internal {
    (bool success,) = BTC_PRECOMPILE.call(abi.encodePacked(BROADCAST_BYTES, signedTx));
    if (!success) revert PrecompileCallFailed();
}
```

Otherwise, it is possible for `SovaBitcoin.broadcastBitcoinTx()` to not revert even when the specified `signedTx` was not broadcasted.

(3) `UBTC.withdraw()` calls the BTC precompile's `VaultSpend` function after burning the caller's UBTC, which broadcasts a bitcoin transaction to transfer BTC on the caller's behalf:

```
bytes memory inputData =
    abi.encode(SovaBitcoin.UBTC_SIGN_TX_BYTES, msg.sender, amount, btcGasLimit, btcBlockHeight, dest);

// This call will set the slot locks for this contract until the slot resolution is done. Then the
// slot updates will either take place or be reverted.
(bool success, bytes memory returndata) = SovaBitcoin.BTC_PRECOMPILE.call(inputData);
if (!success) revert SovaBitcoin.PrecompileCallFailed();
```

The broadcasted transaction should not fail for any reason apart from the caller providing invalid parameters. Otherwise, if the caller's UBTC is burnt but the bitcoin transaction fails, he will lose funds.

(4) When processing deposits from `UBTC.depositBTC()`, the sova-reth client must ensure that `signedTx` is valid (i.e. the correct amount of BTC was transferred to the caller's corresponding bitcoin address) and finalized. This ensures that UBTC is always backed by BTC on the network.

SovaBTC: Response to each issue as follows:

1). After reviewing the current checkSignature I don't think signature verification is needed and provides unnecessary overhead to the deposit flow. If the signature is bad or a double spend this will the caught by the slot locking mechanisms. Essentially the tx will never reach finality on Bitcoin and the next time a user goes to deposit the `pendingDeposit` mapping with be slot reverted back to zero.

2). If signature verification passes (or doesn't exist), and the broadcast fails, then the user or network monitoring system can still broadcast the Bitcoin tx on their own and due to the delay on deposit finality in the execution client (~6 BTC blocks) the deposit can still be successful and minted token still would reflect 1:1 with BTC.

3). With the pending balance PR (`https://github.com/SovaNetwork/contracts/pull/13`) the _burn function will only be called when an outside user triggers the state to be updated. This lazily triggers finality checks on the locked slot to ensure the BTC tx was broadcast and the tx was confirmed.

4). This is correct. If the tx decoding fails, the precompile must fail and thus the tx must fail. The hope is that in the future this validation pattern can be extended to other BTC tx types like taproot spend scripts or ordinal/rune protocol specifications/ metadata.

Functionality related to `checkSignature()` has been removed in PR 15 and PR 85.

MiloTruck: Acknowledged.

4.4 Informational

4.4.1 `UBTC` should inherit `ERC20` instead of `WETH`

Context:

- UBTC.sol#L22
- UBTC.sol#L99-L111

Description: The `UBTC` contract inherits `WETH`:

```
contract UBTC is WETH, IUBTC, Ownable, ReentrancyGuard {
```

However, it overrides the `deposit()` and `withdraw()` functions to always revert:

```
function deposit() public payable override {
    revert("uBTC: must deposit with native BTC");
}

function withdraw(uint256) public pure override {
    revert("uBTC: must use withdraw with destination");
}
```

Looking at Solady's WETH implementation, it is simply an `ERC20` contract with an additional `deposit()` and `withdraw()` function. Therefore, the code would be simpler if `UBTC` inherited `ERC20` directly, instead of `WETH`.

Recommendation: Modify `UBTC` to inherit Solady's `ERC20` instead of `WETH`. The `deposit()` and `withdraw()` functions can then be removed.

SovaBTC: Fixed in PR 9.

MiloTruck: Verified, the recommendation was implemented.

4.4.2 Minor improvements to code and comments

Context: See below.

Description/Recommendation:

1. SovaL1Block.sol#L15 - Typo: "hieght" -> "height".
2. SovaL1Block.sol#L25-L27 - The `version()` function can be declared external.
3. SovaL1Block.sol#L29-L31 - `SYSTEM_ACCOUNT` should be a constant.
4. SovaL1Block.sol#L41-L60 - Both the `setBitcoinBlockData()` and `setBitcoinBlockDataCompact()` functions implement the same logic of setting state variables in the contract; there isn't a need for both functions. Consider removing `setBitcoinBlockDataCompact()`.

SovaBTC: Fixed in PR 7.

MiloTruck: Verified, the recommended fixes have been implemented.