Fiamma BTC Bridge

Audit Report

contact@bitslab.xyz

https://twitter.com/movebit_

Mon Aug 04 2025

Fiamma BTC Bridge Audit Report

1 Executive Summary

1.1 Project Information

Description	A secure and efficient Bitcoin bridge implementation on Aptos blockchain for cross-chain asset transfers.
Type	Bridge
Auditors	MoveBit
Timeline	Tue Jul 29 2025 - Mon Aug 04 2025
Languages	Move
Platform	Aptos
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/fiamma-chain/aptos-contracts/
Commits	208f537a2e43f7600b4c93364398924dba64698e

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
BRI	bridge/sources/bridge.move	9b3f84dde990fcb0f9a201b9a4d98c0069f8a72f
BPE	bridge/sources/btc_peg.move	c13b32004412061434d72e2806f783da94a19a75
LMA1	bridge/sources/lp_manager.move	b4f65d2e4b8d917e1e2a477791b24f9bcf78c923
ERR	btc-light-client/sources/errors.move	d685f49024d883a8fd3e157c6eb67606a0d645fe
BUT	btc-light-client/sources/btc_utils.move	0edf3c9e5e29f5ed504e60c43cbb401e0fead452
CON1	btc-light-client/sources/constants.move	0cbb83286e97f3599d808e3491677186c1de5bc0
BTY	btc-light-client/sources/btc_types.move	307e96459dbac5c53b003212b47845c9e0f76adf
BTV	btc-light-client/sources/btc_tx_verifier.move	46e5f4fd3654a098b591c15dfc271e44dec9085a
BMI	btc-light-client/sources/btc_mirror.move	55cfd64677b0f408645697dfda7b25a60f7adf32

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	12	12	0
Informational	3	3	0
Minor	3	3	0
Medium	4	4	0
Major	0	0	0
Critical	2	2	0

1.4 MoveBit Audit Breakdown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Formal Verification(Optional)

Perform formal verification for key functions with the Move Prover.

(4) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Fiamma to identify any potential issues and vulnerabilities in the source code of the Fiamma BTC Bridge smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 12 issues of varying severity, listed below.

ID	Title	Severity	Status
BMI-1	Get Work in Period Did Not Account for Block Height	Critical	Fixed
BRI-1	Insufficient Verification Leads to Signature Replay Attack	Critical	Fixed
BRI-2	Verify the set parameter passed in	Minor	Fixed
BRI-3	No Max Fee Rate Limit	Minor	Fixed
BRI-4	LP Orders That Cause No One to Consume	Minor	Fixed
BRI-5	Pause/Unpause Burn Should Only Be Called in Opposite State	Informational	Fixed
BRI-6	No Receiver Address Validation	Informational	Fixed
BUT-1	No Check on Siblings Length Allows Merkle Proof Forgery in BTC_Utils::Get_Tx_Merkle_Root	Medium	Fixed

BUT-2	Potential DoS in Parse_Transaction From Large VarInt Medium Low	Medium	Fixed
BUT-3	No Transaction Outputs Index Boundary Check	Medium	Fixed
BUT-4	Out-Of-Bound Read in BTC_Utils.Read_Varint()	Medium	Fixed
BUT-5	Best Practices in Utils	Informational	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Fiamma BTC Bridge Smart Contract :

BTC Bridge Contract

A secure and efficient Bitcoin bridge implementation on Aptos blockchain for cross-chain asset transfers.

Core Components

Bridge Contract (`bridge.move` **)**

- **Mint Function**: Validates Bitcoin deposits and mints tokens
- **Burn Function**: Burns tokens and emits withdrawal events
- **Configuration**: Manages bridge parameters and access control
- **Validation**: Ensures transaction integrity and limits

BTC Peg Token (`btc_peg.move` **)**

- **Token Management**: Handles token minting and burning
- **Balance Tracking**: Maintains user token balances
- **Standard Interface**: Provides standard token operations

Bitcoin Light Client for Aptos

A Move-based Bitcoin light client that mirrors Bitcoin blockchain state on Aptos, enabling trustless Bitcoin block verification.

Features

- **Block Header Tracking**: Maintains a mirror of Bitcoin blockchain state
- **Chain Reorganization**: Handles Bitcoin chain reorgs automatically

- **Difficulty Adjustment**: Supports Bitcoin difficulty retargeting
- **Auto Submission**: Continuous monitoring and block header submission
- **Testnet/Mainnet**: Configurable for both Bitcoin networks

4 Findings

BMI-1 Get Work in Period Did Not Account for Block Height

Severity: Critical

Status: Fixed

Code Location:

btc-light-client/sources/btc_mirror.move#300-304

Descriptions:

Current implementation of `btc_mirror::get_work_in_period` only calculates work per block. It does not account for all blocks in the period. This could make assertion of `new_work > old_work` a false positive in `submit`.

```
fun get_work_in_period(mirror: &BtcMirror, period: u64): u256 {
    let target = *table::borrow_with_default(&mirror.period_to_target, period, &0);
    btc_utils::compute_work_from_target(target)
}

public fun compute_work_from_target(target: u256): u256 {
    let max_target = constants::max_u256();

    // Prevent overflow when target = max_u256
    assert!(target < max_target, errors::invalid_argument());

    ((max_target - target) / (target + 1)) + 1
}
```

Here's implementation of Fiamma's own btc-light-client solidity impl: Fiamma's own btc-light-client solidity

```
function getWorkInPeriod(uint256 period, uint256 height) private view returns (uint256)
{
    uint256 target = periodToTarget[period];
```

```
    uint256 workPerBlock = (2 ** 256 - 1) / target;

    uint256 numBlocks = height - (period * 2016) + 1;
    assert(numBlocks >= 1 && numBlocks <= 2016);

    return numBlocks * workPerBlock;
}
```

The block number is calculated as such because the period is not raw from input but parsed as `newHeight/2016`. Therefore, `numBlocks` is just difference between height and `period * 2016`.

`compute_work_from_target` in Move code is equivalent to `workPerBlock` calculation, but `get_work_in_period` is missing a `numBlocks` multiplication because `height` is not passed to the function.

Suggestion:

Upstream should pass the height to the function, just like in the Solidity impl.

```
public entry fun submit(
        ...
        // The submitted chain segment contains a difficulty retarget
    if (new_period == old_period) {
        // The old canonical chain is past the retarget
-       old_work = get_work_in_period(mirror, old_period);
+       old_work = get_work_in_period(mirror, old_period, mirror.latest_block_height);
    } else {
    ...
    // Check that we have a new heaviest chain
        if (new_period > parent_period) {

            let new_difficulty_bits =
                btc_utils::get_difficulty_bits_from_header(&last_header);

-           let new_work = get_work_in_period(mirror, new_period);
+           let new_work = get_work_in_period(mirror, new_period, new_height);
            assert!(new_work > old_work, errors::insufficient_total_difficulty());
            ...
```

```
}

/// Get work in a specific period (internal version that takes mirror reference)
- fun get_work_in_period(mirror: &BtcMirror, period: u64): u256 {
+ fun get_work_in_period(mirror: &BtcMirror, period: u64, height: u64): u256 {
    let target = *table::borrow_with_default(&mirror.period_to_target, period, &0);
-   btc_utils::compute_work_from_target(target)
+   let work_per_block = btc_utils::compute_work_from_target(target);
+   let num_blocks = height - (period * 2016) + 1;
+   assert!(num_blocks >= 1 && num_blocks <= 2016, errors::incorrect_block_number());
+
+   return num_blocks  * work_per_block ;
}
```

Resolution:

This issue has been fixed. The client has adopted our suggestions.

BRI-1 Insufficient Verification Leads to Signature Replay Attack

Severity: Critical

Status: Fixed

Code Location:

bridge/sources/bridge.move#322-345

Descriptions:

```
let is_valid =
        bls12381::verify_multisignature(
            &aggr_sig, &aggr_public_key, dest_script_hash
        );
    assert!(is_valid, EINVALID_AGGREGATE_SIGNATURE);
```

`dest_script_hash` is not a suitable verification parameter, which will result in signature replay. The core of this problem is that the signature can be reused by the next `mint`, and a malicious administrator can use the content of the previous signature (since the content of the signed message is fixed each time) for the next `mint`, meaning that the signature checking is null and void. (Since `dest_script_hash` doesn't change almost every time).

Suggestion:

```
- let is_valid =
        bls12381::verify_multisignature(
            &aggr_sig, &aggr_public_key, dest_script_hash
        );
+ let is_valid =
        bls12381::verify_multisignature(
            &aggr_sig, &aggr_public_key, tx_id
        );
    assert!(is_valid, EINVALID_AGGREGATE_SIGNATURE);
```

use `tx_id` replace `dest_script_hash` .

Resolution:

This issue has been fixed. The client has adopted our suggestions.

BRI-2 Verify the set parameter passed in

Severity: Minor

Status: Fixed

Code Location:

bridge/sources/bridge.move#189-289;

bridge/sources/lp_manager.move#138-155

Descriptions:

All setter functions in `bridge.move` (e.g., set_max_fee_rate, set_min_confirmations) and the `update_lp_status` function in `lp_manager.move` lack validation to ensure that the new value differs from the existing one. Consequently, these functions may write identical values to storage repeatedly, resulting in unnecessary state transitions and redundant event emissions. This inefficiency can lead to increased gas consumption and inflated event logs, particularly in scenarios where such functions are invoked frequently or programmatically.

Suggestion:

It is recommended to add a check within these functions.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

BRI-3 No Max Fee Rate Limit

Severity: Minor

Status: Fixed

Code Location:

bridge/sources/bridge.move#224-235

Descriptions:

There is a lack of size limit for the parameter `max_fee_rate`, which can be greater than 100%.

```
/// Set maximum fee rate
    public entry fun set_max_fee_rate(owner: &signer, max_fee_rate: u64) acquires
BridgeConfig {
        let config = borrow_global_mut<BridgeConfig>(@bridge_addr);
        assert!(only_owner(config, signer::address_of(owner)), EUNAUTHORIZED);

        let old_max_fee_rate = config.max_fee_rate;
        config.max_fee_rate = max_fee_rate;

        // Emit fee rate update event
        event::emit(
            MaxFeeRateUpdated { old_max_fee_rate, new_max_fee_rate: max_fee_rate }
        );
    }
```

Suggestion:

Add the limit for the parameter `max_fee_rate`.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

BRI-4 LP Orders That Cause No One to Consume

Severity: Minor

Status: Fixed

Code Location:

bridge/sources/bridge.move#391-453

Descriptions:

Due to the lack of constraints on `lp_withdraw.withdraw_amount` and `amount_sats`, no one will be able to consume LP orders. A malicious user can submit a lot of lp that won't be claimed, and while it won't be a big security risk, it will cause some level of dos attack and is an unintended behavior.

```
public entry fun withdraw_by_lp(
        user: &signer,
        withdraw_id: u64,
        btc_address: String,
        receiver_script_hash: vector<u8>,
        receive_min_amount: u64,
        lp_id: u64,
        amount: u64,
        fee_rate: u64
    ) acquires BridgeConfig {
        ...
        // Create LP withdraw record
        let lp_withdraw = LPWithdraw {
            id: withdraw_id,
            withdraw_amount: amount,
            receiver_addr: btc_address,
            receiver_script_hash,
            receive_min_amount,
            fee_rate,
            timestamp: timestamp::now_seconds(),
            lp_id
        };
```

```
    // Store withdraw record
    table::add(&mut config.lp_withdraws, withdraw_id, lp_withdraw);

    // Same as transfer tokens from user to bridge contract
    btc_peg::burn(user_address, amount);

    // Emit event
    event::emit(WithdrawByLP {
        from_address: user_address,
        withdraw_id,
        btc_address,
        fee_rate,
        amount,
        lp_id,
        receive_min_amount
    });
}
```

To ensure valid consumption, the following condition should be enforced:

`receice_min_amount < amount_sats < lp_withdraw.withdraw_amount`.

Suggestion:

Add constraints for these.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

BRI-5 Pause/Unpause Burn Should Only Be Called in Opposite State

Severity: Informational

Status: Fixed

Code Location:

bridge/sources/bridge.move#256-272

Descriptions:

Should only emit `BurnPausedEvent` when switching state. e.g. When already paused, emitting another BurnPausedEvent could cause user confusion.

```
/// Pause burn functionality
public entry fun pause_burns(owner: &signer) acquires BridgeConfig {
    let config = borrow_global_mut<BridgeConfig>(@bridge_addr);
    assert!(only_owner(config, signer::address_of(owner)), EUNAUTHORIZED);

    config.burn_paused = true;
    event::emit(BurnPausedEvent { paused: true });
}

/// Unpause burn functionality
public entry fun unpause_burns(owner: &signer) acquires BridgeConfig {
    let config = borrow_global_mut<BridgeConfig>(@bridge_addr);
    assert!(only_owner(config, signer::address_of(owner)), EUNAUTHORIZED);

    config.burn_paused = false;
    event::emit(BurnPausedEvent { paused: false });
}
```

Suggestion:

Add an assert of `config.burn_paused == false` (or true) in `pause_burns` (or `unpause_burns`).

Resolution:

This issue has been fixed. The client has adopted our suggestions.

BRI-6 No Receiver Address Validation

Severity: Informational

Status: Fixed

Code Location:

bridge/sources/bridge.move#521-531

Descriptions:

No receiver address validation.

```
/// Refund LP withdrawal after timeout
public entry fun refund_lp_withdraw(
    owner: &signer,
    withdraw_id: u64,
    receiver: address
) acquires BridgeConfig {
    let config = borrow_global_mut<BridgeConfig>(@bridge_addr);
    assert!(only_owner(config, signer::address_of(owner)), EUNAUTHORIZED);

    ...
}
```

Suggestion:

Add a check `assert!(receiver != @0x0, EINVALID_ADDRESS);`

Resolution:

This issue has been fixed. The client has adopted our suggestions.

BUT-1 No Check on Siblings Length Allows Merkle Proof Forgery in BTC_Utils::Get_Tx_Merkle_Root

Severity: Medium

Status: Fixed

Code Location:

btc-light-client/sources/btc_utils.move#166-201

Descriptions:

In the `tc_utils.move` function `get_tx_merkle_root`, the length of `siblings` is not checked to match the expected depth of the Merkle Tree given a `tx_index`.

```
/// Recompute the transaction root given a transaction ID, index, and Merkle proof
    public fun get_tx_merkle_root(
        tx_id: vector<u8>, tx_index: u64, siblings: vector<vector<u8>>
    ): vector<u8> {
        assert!(
            vector::length(&tx_id) == constants::tx_id_size(),
            errors::invalid_transaction_id()
        );

        let result = tx_id;
        let num_siblings = vector::length(&siblings);
        let current_index = tx_index;

        for (i in 0..num_siblings) {
            // ...
```

For example, setting `siblings` to empty and `tx_id` will be the function output. `assert!(block_tx_root == tx_root, errors::merkle_root_mismatch());` can be bypassed by setting `tx_root = block_tx_root`.

Suggestion:

The code needs to add a proper length check.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

BUT-2 Potential DoS in Parse_Transaction From Large VarInt Medium Low

Severity: Medium

Status: Fixed

Code Location:

btc-light-client/sources/btc_utils.move#235-244

Descriptions:

In the `tc_utils.move` function, `parse_transaction` parses `raw_tx` for a list of TxIn and TxOut. If VarInt is too large, the for loop could potentially run out of gas, causing DoS.

```
// Read number of inputs (VarInt)
let (num_inputs, new_offset) = read_varint(raw_tx, offset);
offset = new_offset;
// Parse inputs
let inputs = vector::empty<TxIn>();
for (i in 0..num_inputs) {
    let (tx_in, new_offset) = parse_tx_input(raw_tx, offset);
    vector::push_back(&mut inputs, tx_in);
    offset = new_offset;
};

/// Read a VarInt from a byte array
public fun read_varint(data: &vector<u8>, offset: u64): (u64, u64) {
    ...
    } else {
        let value = from_bcs::to_u64(vector::slice(data, offset + 1, offset + 9));
        (value, offset + 9)
    }
}
```

For a sufficiently large VarInt, such as 2^63, we might expect `for (i in 0..num_inputs)` to run out of gas, causing an unexpected revert.

Suggestion:

Add Maximum Size Limit to num_inputs to Prevent Excessive Resource Consumption

Resolution:

This issue has been fixed. The client has adopted our suggestions.

BUT-3 No Transaction Outputs Index Boundary Check

Severity: Medium

Status: Fixed

Code Location:

btc-light-client/sources/btc_utils.move#397-403

Descriptions:

`verify_op_return_output` is missing out of boundary check for transaction `outputs` `vector<TxOut>` . Potential out-of-bound access causing unexpected revert.

```
/// Verify OP_RETURN output contains expected data
fun verify_op_return_output(
    tx: &Transaction, output_index: u64, expected_data: vector<u8>
) {
    let outputs = btc_types::get_outputs(tx);
        // Missing check here for next line
    let tx_out = vector::borrow(outputs, output_index);
    ...
}
```

Suggestion:

Add the following line to increase checks

```
assert!(
    output_index < vector::length(outputs),
    errors::output_index_out_of_bounds()
);
```

Resolution:

This issue has been fixed. The client has adopted our suggestions.

BUT-4 Out-Of-Bound Read in BTC_Utils.Read_Varint()

Severity: Medium

Status: Fixed

Code Location:

btc-light-client/sources/btc_utils.move#520-527

Descriptions:

The check `offset < vector::length(data)` is insufficient. If `pivot ≥ 0xfd`, the end range for reading is `offset + 3` to `offset + 9`. Say `offset == vector::length(data) - 1`, out-of-bound read protection will be triggered. Depending on how other functions call this util function, tx could always revert in edge cases.

```
/// Read a VarInt from a byte array
public fun read_varint(data: &vector<u8>, offset: u64): (u64, u64) {
    assert!(offset < vector::length(data), errors::varint_buffer_too_short());

    let pivot = *vector::borrow(data, offset);

    if (pivot < 0xfd) {
        ((pivot as u64), offset + 1)
    } else if (pivot == 0xfd) {
        let value = from_bcs::to_u16(vector::slice(data, offset + 1, offset + 3));
        ((value as u64), offset + 3)
    } else if (pivot == 0xfe) {
        let value = from_bcs::to_u32(vector::slice(data, offset + 1, offset + 5));
        ((value as u64), offset + 5)
    } else {
        let value = from_bcs::to_u64(vector::slice(data, offset + 1, offset + 9));
        (value, offset + 9)
    }
}
```

Suggestion:

Add boundary checks to ensure there are enough bytes remaining in the vector for each VarInt case.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

BUT-5 Best Practices in Utils

Severity: Informational

Status: Fixed

Code Location:

btc-light-client/sources/btc_utils.move#345-347,295-297

Descriptions:

`constants::max_script_size()` check should be replaced by a length check depending on `script_type`

```
    /// Maximum script size supported
    public fun max_script_size(): u64 {//TODO info 1
        34
    }

// parse_tx_input & parse_tx_output
assert!(
    script_len <= constants::max_script_size(), errors::invalid_script_size()
);
```

`script_type` should be passed in to validate against their respective sizes, instead of the universal max script size. Potentially, someone using P2SH could pass in a script size larger than 23.

```
/// P2SH script size in bytes
public fun p2sh_script_size(): u64 {
    23
}
```

Suggestion:

`script_type` should be passed in to validate against their respective sizes, instead of the universal max script size.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.