Zellic

Prepared for
Bitlayer Labs Ltd.

Prepared by
Mohit Sharma
Avraham Weinstock
Zellic

April 29, 2025

BitVM Bridge
Smart Contract Security Assessment

Contents

About Zellic 4

1. Overview 4
   1.1. Executive Summary 5
   1.2. Goals of the Assessment 5
   1.3. Non-goals and Limitations 5
   1.4. Results 5

2. Introduction 6
   2.1. About BitVM Bridge 7
   2.2. Methodology 7
   2.3. Scope 9
   2.4. Project Overview 10
   2.5. Project Timeline 10

3. Detailed Findings 10
   3.1. Internal key of Taproot outputs is the attesting committee's key 11
   3.2. Unconstrained arguments for `OP_CSV` in `ScriptGenerator` 12

4. Discussion 12
   4.1. Attesting committee is immutable after initialization 13
   4.2. Test suite 13

5. System Design **13**

5.1. Component: Transactions 14

5.2. Component: Contracts 17

6. Assessment Results **17**

6.1. Disclaimer 18

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Bitlayer Labs Ltd. from April 10th to April 22nd, 2025. During this engagement, Zellic reviewed BitVM Bridge's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Does the bridge correctly establish a correspondence between locked BTC and YBTC?
- Does the bridge prevent BTC from being locked without a path to unlock it?
- Are brokers sufficiently disincentivized from producing invalid requests for reimbursement?
- Are watchers sufficiently disincentivized from issuing invalid challenges to brokers?
- Do the Ethereum smart contracts correctly parse data from Bitcoin?
- Do the watcher and broker correctly parse data from the Bitcoin and Ethereum networks?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Correctness of the scripts in the BitVM Groth16 verifier
- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped BitVM Bridge crates, we discovered two findings. No critical issues were found. One finding was of medium impact and the other finding was informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of Bitlayer Labs Ltd. in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
🟥 Critical	0
🟧 High	0
🟨 Medium	1
🟩 Low	0
⬜ Informational	1

2. Introduction

2.1. About BitVM Bridge

Bitlayer Labs Ltd. contributed the following description of BitVM Bridge:

> The BitVM Bridge represents a groundbreaking advancement in Bitcoin bridging technology, developed by Bitlayer and powered by the innovative BitVM smart contract. As the first step toward the realization of the Bitlayer rollup, BitVM Bridge offers a secure and efficient mechanism for transferring Bitcoin (BTC) into the Bitlayer ecosystem. This integration allows Bitcoin to be actively utilized within the BTCFi ecosystem, enabling seamless trading and interaction with decentralized finance (DeFi) applications.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the crates.

Architecture risks. This encompasses potential hazards originating from the blueprint of a system, which involves its core validation mechanism and other architecturally significant constituents influencing the system's fundamental security attributes, presumptions, trust mode, and design.

Arithmetic issues. This includes but is not limited to integer overflows and underflows, floating-point associativity issues, loss of precision, and unfavorable integer rounding.

Implementation risks. This encompasses risks linked to translating a system's specification into practical code. Constructing a custom system involves developing intricate on-chain and off-chain elements while accommodating the idiosyncrasies and challenges presented by distinct programming languages, frameworks, and execution environments.

Availability. Denial-of-service attacks are another leading issue in custom systems. Issues including but not limited to unhandled panics, unbounded computations, and incorrect error handling can potentially lead to consensus failures.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped crates itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

BitVM Bridge Crates

Types	Rust, Solidity
Platforms	Bitcoin, EVM-compatible

Target	bitvm-bridge
Repository	https://github.com/bitlayer-org/bitvm-bridge ↗
Version	`8828560e893e88ae067dac9e1d05d4960130b33f`

Programs	`bridge-defender/`
	`transactions/`
	`primitives/`

Target	bitvm-bridge-contracts
Repository	https://github.com/bitlayer-org/bitvm-bridge-contracts ↗
Version	`157fa078fd8dd2c590b2c00252f78abe01ff3fbb`

Programs	`AttestingCommittee.sol`
	`BitlayerBridgeV2.sol`
	`YBTC.sol`
	`lib/AdvancedBitcoinTx.sol`
	`lib/Pegin.sol`
	`lib/Pegout.sol`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 2.7 person-weeks. The assessment was conducted by two consultants over the course of 1.8 calendar weeks.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io

Chad McDonald
Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Mohit Sharma
Engineer
mohit@zellic.io

Avraham Weinstock
Engineer
avi@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

April 10, 2025	Kick-off call
April 10, 2025	Start of primary review period
April 22, 2025	End of primary review period

3. Detailed Findings

3.1. Internal key of Taproot outputs is the attesting committee's key

Target	crates/protocol/transactions		
Category	Coding Mistakes	**Severity**	High
Likelihood	Low	**Impact**	Medium

Description

All of the Taproot outputs currently use the attesting committee's key as the internal key, allowing the attesting key to unilaterally spend them.

Impact

For the Pegin transaction, there is no additional impact as a script that requires only the attesting committee's signature is already present as a leaf (though that leaf could be removed and the internal key used instead, as an optimization). For the Kickoff and Assert transactions, the attesting committee is in a position to bypass the timelocks and immediately claim the broker's deposit.

Recommendations

Use a public key with no corresponding private key (e.g., `lift_x(0x50929b74c1a04954b78b4b6035e97a5e078a5a0f28ec96d547bfee9ace803ac0)` from BIP-341 ↗) as the internal key for all Taproot outputs, except possibly the Pegin transaction's output. If using the attesting committee's key for the Pegin transaction's output's internal key, remove the leaf containing the script that checks the attesting committee's key, and sign transactions that would use that leaf with the internal key.

Remediation

This issue has been acknowledged by Bitlayer Labs Ltd., and a fix was implemented in commit `cbf97474` ↗.

3.2. Unconstrained arguments for `OP_CSV` in `ScriptGenerator`

Target	crates/protocol/transactions		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	Low	**Impact**	Informational

Description

The following methods in `ScriptGenerator` utilize the `OP_CSV` opcode: `add_timelock_script` and `p2wsh_or_timelock`.

Both of these methods accept `block_count` as an argument of type `i64` but do not take into account the opcode's treatment of specific values in the argument. For example, `OP_CSV` can only handle a timelock of up to 65,535 blocks. This upper limit is not enforced by `ScriptGenerator`. Additionally, the 22nd bit of the input is used as a flag to distinguish between lock time being specified in blocks as opposed to seconds.

Impact

The affected methods are used for adding timelocks during the construction of the Kickoff and Assert transactions. The wait time that is used as an input for `OP_CSV` is derived from protocol parameters. The transactions are built independently by every member of the attestation committee, and therefore, as long as the initial parameters are configured correctly and at least one signer remains honest as per the protocol's security model, the flaw is not exploitable. The lack of constraints here does however allow for invalid protocol parameters to be valid, and we recommend adding them as a sanity check.

Recommendations

Add an upper limit to `block_count` in `ScriptGenerator::add_timelock_script` and `ScriptGenerator::p2wsh_or_timelock`.

Remediation

This issue has been acknowledged by Bitlayer Labs Ltd., and a fix was implemented in commit `ba4a5d89` ↗.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Attesting committee is immutable after initialization

The list of members tracked by `AttestingCommittee` is immutable after initialization since the contract does not expose any public functions for the admin to grant or revoke the `ATTESTER_ROLE` role to addresses.

It is also to be noted that the `updateThreshold` function does not impose a lower limit on the threshold. This poses a centralization risk as a compromised admin can set the threshold to 1 (0 is not possible since it is checked for in the `verifySignatures` function) and use a single malicious attester to act as the committee. This risk could be mitigated by also having a minimum allowable threshold (e.g. 50%).

4.2. Test suite

The `bridge-defender` contains integration tests that test that the constructed Bitcoin transactions behave as expected against an in-process mock Bitcoin executor, with both a mock zero-knowledge proof verifier and the actual zero-knowledge proof verifier.

The Solidity contracts have a hardhat test suite that tests the contracts against provided data.

5. System Design

This provides a description of the high-level components of the system and how they interact, including details like a function's externally controllable inputs and how an attacker could leverage each input to cause harm or which invariants or constraints of the system are critical and must always be upheld.

Not all components in the audit scope may have been modeled. The absence of a component in this section does not necessarily suggest that it is safe.

5.1. Component: Transactions

Description

The Bitcoin side of the bridge consists of a set of transactions that lock BTC for use as collateral for minting YBTC.

The following parties are involved in the bridging process:

- Users initiate the processes of locking BTC to mint YBTC and burning YBTC to unlock BTC.
- The attesting committee acts as a covenant emulation committee, presigning the set of transactions to ensure that only the specific set of transactions are valid. The attesting committee uses short-term keys for signing the Bitcoin transactions as Bitcoin transactions and long-term keys for signing the Bitcoin transactions for the Ethereum contracts to recognize them.
- Brokers service requests to unlock BTC by providing BTC on burn events and being reimbursed by the unlocked BTC.
- Watchers challenge attempts by brokers to redeem BTC that does not have a valid burn proof.
- The security council can claim Pegin deposits in an emergency and redistribute funds from slashing.

Transaction: Pegin

The Pegin transaction is created by a user to deposit BTC into the bridge. It can have multiple inputs funding it, multiple outputs to the Pegin Taproot script address (with potentially different amounts), an `OP_RETURN` output indicating which Ethereum address should receive the bridged YBTC, and an optional change address.

The Pegin Taproot script requires a signature from either the attesting committee or the security council.

Transaction: Pegout

The Pegout transaction is created by a user burning YBTC to withdraw BTC from the bridge, who partially signs it with the requested amount. A broker, on observing the burn event from Ethereum, provides the funding inputs, its optional change output, and an `OP_RETURN` output with the Pegin transaction's ID and output index as well as the broker's address into which the funds will be unlocked.

Transaction: Kickoff

The Kickoff transaction is created by a broker to be reimbursed after funding a Pegout transaction. Its inputs are a deposit from the broker that sum to at least `challenge_amount + dust_amount + normal_tx_gas + guarantee_amount`, both covering fees and providing a disincentive to the brokers for submitting invalid claims. Its outputs are a connector with `dust_amount` (which requires the broker's signature) that ensures that at most one of the HappyTake and Challenge transactions are published, a deposit of at least `challenge_amount + guarantee_amount` is spendable by the assert Taproot script, and an `OP_RETURN` output contains the transaction ID and output of the Pegin transaction to be unlocked (the same data present in the Pegout transaction's `OP_RETURN` output).

The assert Taproot script has three paths:

1. A timelock path locked for `challenge_window + assert_window` blocks that requires a signature from the attesting committee (used by the AssertTimeout transaction to slash the broker if the broker fails to reveal commitments)
2. A path that requires signatures from both the broker and the attesting committee (used by the Assert transaction if the broker publishes the commitments in response to a challenge from a watcher)
3. A timelock path locked for `challenge_window` blocks that requires a signature from the attesting committee (used by the HappyTake transaction if no watcher issues a challenge)

Transaction: HappyTake

The HappyTake transaction can be published no sooner than `challenge_window` blocks after the broker submits the Kickoff transaction if no watcher submits the Challenge transaction. Its inputs are an output from the Pegin transaction that provides the funds to unlock from the user's deposit, the connector output from the Kickoff transaction (which prevents the HappyTake transaction from being published if the Challenge transaction was published), and the third path of the Kickoff transaction's Taproot output, which enforces the timelock and refunds the broker's Kickoff deposit. Its output is spendable by the broker.

Transaction: Challenge

The Challenge transaction is published by a watcher in response to an invalid Kickoff. Its inputs are the Kickoff transaction's connector output (preventing the HappyTake transaction from being subsequently published) and deposits from the watcher adding up to at least `challenge_amount + normal_tx_gas` (which disincentivize spurious challenges and subsidize the broker's transaction fees). Its outputs are the `challenge_amount` spendable by the broker and optionally a change output for the watcher.

Transaction: Assert

The Assert transactions are a sequence of transactions from the broker, which publish the scripts from the BitVM verifier that correspond to the broker's execution of the zero-knowledge proof that the YBTC was burned. They thread through a connector input-output pair, starting from the Kickoff transaction's connector, with a lock Taproot output that propagates the deposit value (minus the `assert_tx_gas` and `dust_amount` for the assert Taproot) and an assert Taproot output with a `dust_amount` value (with scripts provided by the BitVM verifier for the Disprove transaction to be published if the zero-knowledge proof does not verify).

The lock Taproot output allows for either the broker to publish the next Assert transaction or for an AssertTimeout transaction to be published if the next Assert transaction is not published after `assert_window` blocks (or `challenge_window + assert_window`, from the second path of the Kickoff transaction's Taproot for the first Assert transaction).

Transaction: AssertTimeout

The AssertTimeout transactions are a sequence of transactions that slash the broker if they fail to publish the corresponding Assert transaction within the specified time frame. They take the lock Taproot output that the corresponding Assert transaction would use as input and have one output sending `burn_amount` of the value to the security council and the remainder of the value (minus `normal_tx_gas`) to the watcher.

Transaction: Disprove

The Disprove transaction is published by the watcher to provide evidence that a broker provided an invalid zero-knowledge proof. Its inputs are the outputs from the final Assert transaction, requiring the watcher to provide a disproof in the format checked by the BitVM verifier and providing the value of the broker's deposit. Its outputs are a `burn_amount` sent to the security council and the remainder of the input value minus `slash_tx_gas` sent to the watcher (which incentivizes the watchers to submit valid challenges).

Transaction: UnhappyTake

The UnhappyTake transaction is published by a broker when a watcher has issued a challenge but the challenge was spurious (and thus the watcher could not submit a Disprove transaction before the time-out). Its inputs are the Pegin transaction's output being unlocked (which provides the user's deposit to reimburse the broker) and the Assert transaction's outputs (which enforce the timelock and provide the watcher's deposit to slash the watcher for the invalid challenge). Its output sends the value minus `normal_tx_gas` to the broker.

5.2. Component: Contracts

Description

The EVM side of the bridge consists of a set of contracts that can provably verify locked BTC and utilize it as collateral for minting YBTC.

Contract: AttestingCommittee

This contract keeps track of the members of the attesting committee on the EVM side of BitLayer.

The `verifySignatures` function is responsible for

- decoding and verifying a variable length list of signatures on the same message hash
- verifying that each signature has been generated by a unique member of the attesting committee
- verifying that the number of signatures exceeds the preconfigured threshold of signers

Contract: BitLayerBridgeV2

This is the main bridge contract on the EVM side. It handles the EVM side of peg-in and peg-out operations by minting and burning YBTC after verifying appropriate proofs.

Pegin

The peg-in process uses the `mint` function, which

- validates both the input and output vectors of the Bitcoin transaction,
- implements replay protection using TX hash as the key,
- verifies attester committee signatures for the transaction hash,
- uses a dummy relay for the testnet (this should be switched out for production mode), and
- mints the correct amount of tokens on the YBTC contract.

Pegout

The peg-out process uses the `burn` function, which

- validates that the referenced peg-in UTXO is unspent and matches the burn `amount`,
- validates the presigned peg-out Bitcoin transaction's inputs and outputs,
- implements replay protection for the peg-out request, and
- transfers the YBTC tokens from the user to this contract and burns them thereafter.

6. Assessment Results

At the time of our assessment, the reviewed code was not deployed to the Bitcoin or Ethereum mainnets.

During our assessment on the scoped BitVM Bridge crates, we discovered two findings. No critical issues were found. One finding was of medium impact and the other finding was informational in nature.

6.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.