

THORChain Bifrost UTXO Client

Application Security Assessment

Contents

About Zellic	3
<hr/>	
1. Overview	3
1.1. Executive Summary	4
1.2. Goals of the Assessment	4
1.3. Non-goals and Limitations	4
1.4. Results	4
<hr/>	
2. Introduction	5
2.1. About THORChain Bifrost UTXO Client	6
2.2. Methodology	6
2.3. Scope	8
2.4. Project Overview	9
2.5. Project Timeline	9
<hr/>	
3. Discussion	9
3.1. Potential insolvency issues arising from losses due to chain reorgs	10
<hr/>	
4. System Design	10
4.1. Bifrost service	11
<hr/>	
5. Assessment Results	15
5.1. Disclaimer	16

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](#) ⁷ 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 ⁷ and follow [@zellic_io](#) ⁷ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ⁷.



1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for THORChain from January 6th to January 14th, 2025. During this engagement, Zellic reviewed THORChain Bifrost UTXO Client'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:

- Are there any issues with transaction parsing for inbound transactions in the chain client?
 - Are there any issues with transaction signing in the Bifrost module?
 - Is it possible for an attacker to insert fake transactions through the observer?
-

1.3. Non-goals and Limitations

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

- Front-end components
- Infrastructure relating to the project
- Key custody

This audit only conducted a security evaluation for the UTXO chain client in the Bifrost module. The chain client itself only contains as much logic as required to connect to the respective external chains. Any outbound transactions emitted by THORChain are perceived as trusted by the Bifrost module. Therefore, any security issues arising from THORChain itself would be outside the scope of this audit. Moreover, the RPC node used by the observer to scan external chains for transactions is also considered a trusted resource.






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 THORChain Bifrost UTXO Client modules, there were no security vulnerabilities discovered.

Zellic recorded its notes and observations from the assessment for the benefit of THORChain in the Discussion section ([3. 7](#)).

Breakdown of Finding Impacts

Impact Level	Count
 Critical	0
 High	0
 Medium	0
 Low	0
 Informational	0

2. Introduction

2.1. About THORChain Bifrost UTXO Client

THORChain contributed the following description of THORChain Bifrost UTXO Client:

THORChain is secured by its native token, RUNE, which deterministically accrues value as more assets are deposited into the network.

Anyone can use THORChain to swap native assets between any supported chains or deposit their assets to earn yield from swaps.

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 modules.

Nondeterminism. Nondeterminism is a leading class of security issues on Cosmos. It can lead to consensus failure and blockchain halts. This includes but is not limited to vectors like wall-clock times, map iteration, and other sources of undefined behavior (UB) in Go.

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

Complex integration risks. Several high-profile exploits have been the result of unintended consequences when interacting with the broader ecosystem, such as via IBC (Inter-Blockchain Communication Protocol). Zellic will review the project's potential external interactions and summarize the associated risks. If applicable, we will also examine any IBC interactions against the ICS Specification Standard to look for inconsistencies, flaws, and vulnerabilities.

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 modules itself. These observations — found in the Discussion ([3.7](#)) 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:

THORChain Bifrost UTXO Client Modules

Type	Go
Platform	THORChain
Target	UTXO Chain Client
Repository	https://gitlab.com/thorchain/thornode
Version	c04638cf56ecafa30ec5d1571f6470f060b5ad23
Programs	bitcoin.go rpc/rpc.go wire_gen.go signable_gen.go client.go bitcoincash.go dogecoin.go wire.tpl generate.go client_internal.go litecoin.go signer.go helpers.go signable.tpl signer_internal.go

2.4. Project Overview

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

↗ **Frank Bachman**
↗ Engineer
frank@zellic.io ↗

↗ **Junyi Wang**
↗ Engineer
junyi@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

January 6, 2025	Start of primary review period
------------------------	--------------------------------

January 14, 2025	End of primary review period
-------------------------	------------------------------

3. 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.

3.1. Potential insolvency issues arising from losses due to chain reorgs

Every time the Bifrost service detects a new block at a previous height it has seen before, it scans the newly discovered block for previously reported transactions. If it finds a transaction that is missing, it means the transaction has been reorged out. The missing transaction is reported to THORChain as an ErrataTx. Any state changes associated with the transaction are reversed.

The required confirmation count for a block is derived from the total transaction value of the block. This is designed such that a malicious reorg is unlikely to happen, as the cost of the reorg would exceed the sum of value gained. However, losses arising due to reorgs would be socialized across LPs, potentially causing the backing vaults to be insolvent.

4. System Design

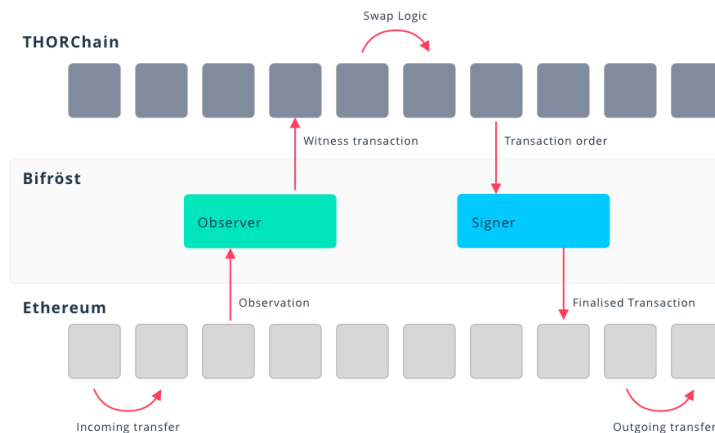
This provides a description of the high-level components of the system and how they interact, including details like a function's externally controllable inputs and how an attacker could leverage each input to cause harm or which invariants or constraints of the system are critical and must always be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

4.1. Bifrost service

All Thorchain nodes run a Bifrost service, which is primarily responsible for interacting with external chains. This involves both inbound and outbound transactions from supported external chains. The Bifrost service for each chain client consists of two components:

1. **Observer.** This scans blocks in external chains to watch for vault addresses. Any inbound transactions discovered by the observer are converted into THORChain witness transactions to be sent to the THORChain L1 layer.
2. **Signer.** The THORChain L1 layer processes the inbound transaction to return a remittance request to the signer. This outbound request is sent to a chain client specific to the external chain, which then signs and broadcasts the transaction.



THORChain Bifrost uses Asgard vaults, which are used to both receive and send assets from external chains. These Asgard vaults use a threshold signature scheme (TSS) requiring signatures from multiple nodes to sign a transaction. Each Asgard vault is limited to a maximum of 20 nodes, and it scales based on the number of nodes in the network. The observer looks for UTXO spending into these vaults and sends witness transactions to THORChain.

```
type Tx struct {
    ID          TxID    `json:"id"`
    Chain       Chain   `json:"chain"`
    FromAddress Address `json:"from_address"`
    ToAddress   Address `json:"to_address"`
    Coins       Coins   `json:"coins"`
    Gas         Gas     `json:"gas"`
    Memo        string  `json:"memo"`
}
```

The Bifrost module also maintains an LRU cache to avoid resigning the same transaction.

UTXO Chain Client

The UTXO chain client allows the Bifrost module to create a new client through `NewClient`:

```
func NewClient(
    thorKeys *thorclient.Keys,
    cfg config.BifrostChainConfiguration,
    server *gotss.TssServer,
    bridge thorclient.ThorchainBridge,
    m *metrics.Metrics,
) (*Client, error) {
    // verify the chain is supported
    supported := map[common.Chain]bool{
        common.DOGChain: true,
        common.BCHChain: true,
        common.LTCChain: true,
        common.BTCChain: true,
    }
    if !supported[cfg.ChainID] {
        return nil, fmt.Errorf("unsupported utxo chain: %s", cfg.ChainID)
    }
    // create rpc client
    rpcClient, err := rpc.NewClient(cfg.RPCHost, cfg.UserName, cfg.Password,
        cfg.MaxRPCRetries, logger)
    // node key setup
    tssKeysign, err := tss.NewKeySign(server, bridge)
```

It allows the creation of a chain client for any of the supported UTXO chains. This also sets up the RPC connection with the respective external chain to scan and extract transactions, and it registers the TSS key to sign a transaction for the Asgard vault assigned to the node.

The client parses the raw transaction to check that at least one output is spent towards an Asgard vault and that another output has an OP_RETURN. The resulting witness transaction is then sent to

THORChain.

```
func (c *Client) getTxIn(tx *btcjson.TxRawResult, height int64, isMemPool bool,
    vinZeroTx map[string]*btcjson.TxRawResult) (types.TxInItem, error) {
    [...]
    if c.isAsgardAddress(toAddr) {
        // only inbound UTXO need to be validated against multi-sig
        if !c.isValidUTXO(output.ScriptPubKey.Hex) {
            return types.TxInItem{}, fmt.Errorf("invalid utxo")
        }
    }
    amount, err := btcutil.NewAmount(output.Value)
    if err != nil {
        return types.TxInItem{}, fmt.Errorf("fail to parse float64: %w", err)
    }
    amt := uint64(amount.ToUnit(btcutil.AmountSatoshi))

    gas, err := c.getGas(tx)
    if err != nil {
        return types.TxInItem{}, fmt.Errorf("fail to get gas from tx: %w", err)
    }
    return types.TxInItem{
        BlockHeight: height,
        Tx:          tx.Txid,
        Sender:      sender,
        To:          toAddr,
        Coins: common.Coins{
            common.NewCoin(c.cfg.ChainID.GetGasAsset(), cosmos.NewUint(amt)),
        },
        Memo: memo,
        Gas:  gas,
    }
}
```

Inbound transactions are “conf-counted” by the chain client and sent to THORChain along with the required confirmation count. THORChain does not process these transactions until the required block height is reached (i.e., for a required confirmation count of one, it will immediately process the transaction). The UTXO chain client computes the required confirmations in `getBlockRequiredConfirmation`:

```
// getBlockRequiredConfirmation find out how many confirmation the given txIn
// need to have before it can be send to THORChain
func (c *Client) getBlockRequiredConfirmation(txIn types.TxIn, height int64)
(int64, error) {
    totalTxValue := txIn.GetTotalTransactionValue(c.cfg.ChainID.GetGasAsset(),
    c.asgardAddresses)
    totalFeeAndSubsidy, err := c.getCoinbaseValue(height)
```

```

    if err != nil {
        c.log.Err(err).Msgf("fail to get coinbase value")
    }
    confMul, err := utxo.GetConfMulBasisPoint(c.GetChain().String(), c.bridge)
    [...]
    confValue := common.GetUncappedShare(confMul,
        cosmos.NewUint(constants.MaxBasisPts),
        cosmos.SafeUintFromInt64(totalFeeAndSubsidy))
    confirm := totalTxValue.Quo(confValue).Uint64()
    confirm, err = utxo.MaxConfAdjustment(confirm, c.GetChain().String(),
        c.bridge)
    if err != nil {
        c.log.Err(err).Msgf("fail to get max conf value adjustment for %s",
            c.GetChain().String())
    }
    if confirm < c.cfg.MinConfirmations {
        confirm = c.cfg.MinConfirmations
    }
    [...]
    return int64(confirm), nil
}

```

The confirmation count is calculated by dividing the total transaction value of the block (spending on Asgard vaults) to the derived confirmation value based on the Coinbase value and chain-specific multiplier. If this is lower than the configured minimum for the chain, the configured value is used instead.

Signer

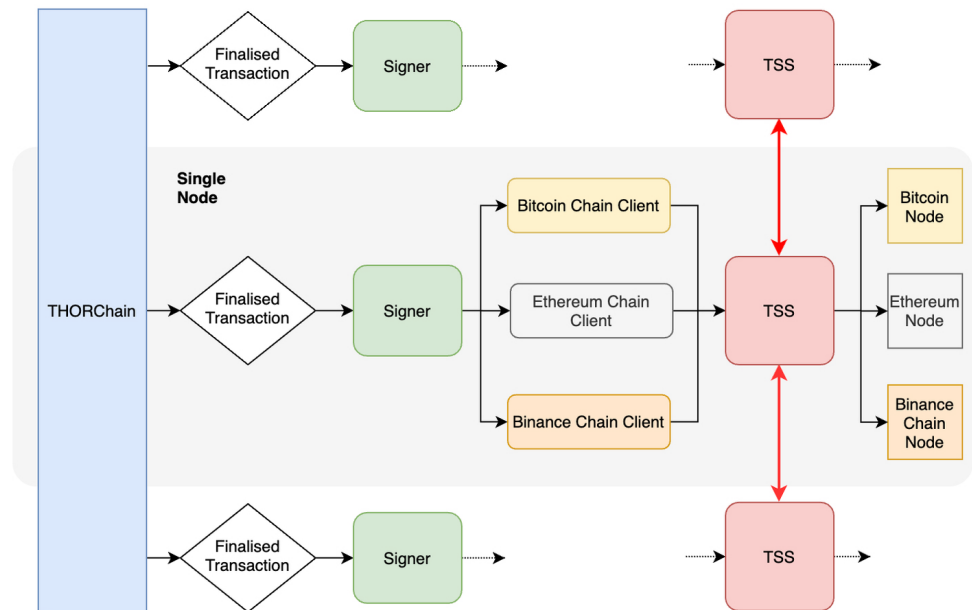
Once there is consensus among the majority nodes and the transaction is finalized, it is sent to the chain client for that chain. The signing process is invoked through the SignTx method in the Chain-Client interface.

```

// ChainClient is the interface for chain clients.
type ChainClient interface {
    [...]
    // SignTx returns the signed transaction.
    SignTx(tx types.TxOutItem, height int64) ([]byte, []byte, *types.TxInItem,
        error)
}

```

The signer uses the TSS module to process the key signing for the assigned Asgard vault. Once signed, the final transaction is broadcasted to the external chain.



The signer for the UTXO client specifically performs the following operations:

- It verifies that the transaction is for the correct chain.
- It looks up the signer cache to verify that the transaction has not already been signed.
- It acquires a vault-specific lock to prevent concurrent signing.
- It decodes and verifies the recipient address (ToAddress), retrieves the source script for inputs, and processes any existing checkpoint to handle partially signed transactions.
- If no checkpoint exists, it builds the transaction from scratch, serializes it, and prepares inputs for signing.
- Each input is signed in parallel through the chain-specific signing function for the UTXO chain. This uses the TSS signing module to sign the inputs for the Asgard vaults.
- Once all inputs are signed, the transaction is serialized back into its UTXO-specific format, and its size and gas usage are calculated.
- It then constructs the final signed transaction, a checkpoint (for recovery on failure), and observation details (like gas used and outbound amount) to broadcast or handle the transaction further.

If the transaction signing is successful, the Bifrost module broadcasts it to the external chain to be executed.

5. Assessment Results

At the time of our assessment, the reviewed code was not deployed.

During our assessment on the scoped THORChain Bifrost UTXO Client modules, there were no security vulnerabilities discovered.

5.1. 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.