coinspect

You build, we defend.

vechain

Blockchain for Our Better World

Source Code Audit

VeChainThor Galactica

May 2025

VeChainThor Galactica
Source Code Audit

Version: v250505 | Prepared for: Vechain Foundation | May 2025

Security Assessment

1. Executive Summary

In **April 2025**, **VeChain Foundation** engaged Coinspect to perform a Source Code Audit of the VeChainThor Blockchain Galactica Update. The objective of the project was to evaluate the security of the update.

`VeChainThor` is an EVM-compatible blockchain focused on being sustainable by offering low-cost transactions; a two-token model with one token specifically for gas; delegations allowing a user to pay for the fees of another and *clauses*, transactions with multiple operations that are either successful as a group or outright rejected.

✓ Solved	⚠ Caution Advised	✗ Resolution Pending
High 0	High 0	High 0
Medium 1	Medium 0	Medium 0
Low 0	Low 1	Low 0
No Risk 3	No Risk 0	No Risk 0
Total **4**	Total **1**	Total **0**

This report contains 4 issues, one of which is of medium severity and details how a blocklisted account can still use `VTHO` to sponsor transactions. The rest of the findings are of `low` severity or informational. The VeChain team has fixed the medium severity issue while acknowledging the rest.

2. Summary of Findings

This section provides a concise overview of all the findings in the report grouped by remediation status and sorted by estimated total risk.

2.1 Findings where caution is advised

Issues with risk in this list have been addressed to some extent but not fully mitigated. Any future changes to the codebase should be carefully evaluated to avoid exacerbating these issues or increasing their probability.

Findings with a risk of `None` pose no threat, but document an implicit assumption which must be taken into account. Once acknowledged, these are considered solved.

Id	Title	Risk
VCT-002	HTTP Client has no timeout	Low

2.2 Solved issues & recommendations

These issues have been fully fixed or represent recommendations that could improve the long-term security posture of the project.

Id	Title	Risk
VCT-001	Blocklisted account can use VTHO	Medium
VCT-003	A hash result of zero panics the chain	None
VCT-004	A zero-gas transaction would panic the chain	None
VCT-005	A well-positioned attacker can manipulate `COM()` vote of block	None

3. Scope

The review started on April 7th 2025 and was scheduled for 3 weeks.

The scope was set to be the repository at `https://github.com/vechain/thor` at commit `dda032fae6bebdb6e8302edfc7d0662f20c468d9`, the `HEAD` of the branch `release/galactica` when the review started. The review was a differential review dealing only with the changes between the `master` branch and the `release/galactica` branch at the point when the review started. The `HEAD` of `master` at the time was `855ae30a4ddccf6cbabc8a184678cbbafd265fb9`.

Thus, the scope was set as the diff given by `git diff 855ae30a4ddccf6cbabc8a184678cbbafd265fb9 dda032fae6bebdb6e8302edfc7d0662f20c468d9`.

3.1 Fixes review

The fixes review analyzed commit `35937fcfe79afbf4f1ecceb93746e2f7378b177b` for the fix to `VCT-001`. The rest of the issues were `Acknowledged` by the VeChainThor team.

4. Assessment

`VeChainThor` is a L1-blockchain with a proof-of-authority consensus mechanism. There are 101 validators that are identified by `VeChain`.

The specific changes in scope for this review were related to the `Galactica` hardfork, which introduced a new EIP-1559-like fee market to VeChain. The full specification is described in VIP-251. This introduces a `BaseFee` as a block header field and changes the rewards for block producers: the proof-of-work of transactions is now ignored, and the full reward (priority fee) of transactions is sent to the miners. The `BaseFee` is burned, as in Ethereum.

The `diff` also includes a new envelope format for transactions. The new format is in-use already for new VIP-251 transactions, but it allows new transactions types to be introduced without breaking changes. The specification is in VIP-252.

Another relevant VIP is VIP-242. These are changes made to maintain EVM-compatibility and allow any contract written for Ethereum to run on `VeChainThor`: it introduces the `BASEFEE` and `PUSH0` opcodes, rejection of contracts starting with `0xEF` and changes to the gas cost of the `alt_bn128` and `modexp` precompiled contracts.

Lastly, `txClauseIndex()` and `txClauseCount()` were added to the `Extension` contract (VIP-250).

For the review, Coinspect considered the changes related to `VIP-251` as the most critical: the `VeChainThor` project differs from Ethereum gas handling due to its dual-token model with a token (`VETHO`) dedicated exclusively to paying for gas. Coinspect looked for potential logic bugs stemming from this difference, as well as implementation bugs such as integer overflows.

`VIP-252` was also relevant in the threat model: Coinspect look specially for tampering of transactions due to insecure signatures, as well as potential denial-of-service attacks due to the new serialization logic.

`VIP-242` was reviewed for Ethereum compatibility and potential differences that would lead to vulnerabilities.

`VIP-250` is a smaller change that adds only two `view` functions to a `Solidity` contract. Nevertheless, the calls are dispatched to native code, so Coinspect reviewed is looking for potential panics or mismatches between the expectations of `solidity` and `go` code.

The implementation of the Galactica features involved substantial refactoring, especially concerning transaction representation and serialization. While drawing clear inspiration from Ethereum standards like EIP-1559 and EIP-2718, the adaptation to `VeChainThor`'s unique architecture (dual-token, PoA) required careful integration. The review evaluated the correctness of these adaptations, particularly where `VeChainThor`'s implementation deviates from or interacts with existing mechanisms like fee delegation or the `VTHO` economy.

Most of the other features of VeChain closely mirror those of Ethereum. One important distinction is in the replay and reordering protection of transactions. While Ethereum uses nonces to prevent replay and arbitrary reordering, VeChain uses a `DependsOn` field to prevent reordering, and rejects transactions with the same `txid` to prevent replay. Users should be aware of this difference, both because transactions without a `DependsOn` field are susceptible to be reordered by a block producer, and also because it leads to slight different calculations of contract addresses via `CREATE` and `CREATE2` opcodes in certain situations.

Lastly, Coinspect noted that API protections where missing in the project. The risk of an attacker exploiting this is documented in `VCT-006`. Node operators are encouraged to expose their node to users via a reverse proxy or implement other strategies that allow them to set rate limits.

4.1 Security assumptions

- At least 2/3+1 of block producers are assumed non-byzantine and available.

5. Detailed Findings

VCT-001

Blocklisted account can use VTHO

Status
Solved

Resolution
Fixed

Risk
Medium

Impact
Medium

Likelihood
Medium

Location

`thor/consensus/validator.go`

Description

A blocklisted attacker can still use their `VTHO` tokens as the `delegator` of a secondary account, partially bypassing `VeChainThor`'s consensus-level blocklist.

While current validations check that the `origin` of a transaction is not blocklisted, a blocklisted user can partially bypass this validation and still use their `VTHO` by sponsoring another accounts transaction.

The attacker would have to:

1. Have a blocklisted account with `VTHO`
2. Create a secondary account
3. Sponsor the secondary account with the blocklisted account

The root cause is in the `validateBlockBody` method of the `validator` package:

```
for _, tr := range txs {
    origin, err := tr.Origin()
    if err != nil {
        return consensusError(fmt.Sprintf("tx signer unavailable: %v",
err))
    }

    if header.Number() >= c.forkConfig.BLOCKLIST &&
thor.IsOriginBlocked(origin) {
        return consensusError(fmt.Sprintf("tx origin blocked got
packed: %v", origin))
    }
```

Recommendation

Check that the `delegator` of the transaction is not a blocklisted account.

Status

Fixed in commit `35937fcfe79afbf4f1ecceb93746e2f7378b177b`. Each transaction's delegator is now checked against the blocklist and rejected if present. Note the fix assumes research has been conducted to make sure that no transaction of this characteristic was ever present in the chain.

VCT-002

HTTP Client has no timeout

Status
Caution Advised

Resolution
Acknowledged

Risk
Low

Impact
Low

Likelihood
Low

Location

`thor/thorclient/thorclient.go`

Description

The `thorclient::Client::New()` uses a a `http.DefaultClient` as the underlying HTTP client. Golang's `http.DefaultClient` has no timeout. This means users that use `New()` are at risk of having their connection to services hang if the service is non-responding.

Recommendation

Set a timeout in the `http.DefaultClient` before wrapping it in the `thorclient::Client::Client` struct.

Status

Acknowledged.

VCT-003

A hash result of zero panics the chain

Status
Solved

Resolution
Acknowledged

Risk
None

Impact
Recommendation

Likelihood
-

Location

thor/thorclient/thorclient.go

Description

If the Blake2b digest used in the evaluateWork function to calculate the work done on a transaction results in the zero-hash, the node will crash due to division by zero on big integers. This issue is only informational, as the likelihood of a correct Blake2b resulting in the zero-hash is negligible. Nevertheless, a check should be added to guard against potentially faulty-implementations and for theoretical correctness.

The root cause of the issue can be seen in tx_legacy.go, in the line r.Div(math.MaxBig256, r). If r == 0, the BigInt division will panic.

```
func (t *legacyTransaction) evaluateWork(origin thor.Address)
func(nonce uint64) *big.Int {
        hashWithoutNonce := t.hashWithoutNonce(origin)

        return func(nonce uint64) *big.Int {
                var nonceBytes [8]byte
                binary.BigEndian.PutUint64(nonceBytes[:], nonce)
                hash := thor.Blake2b(hashWithoutNonce[:],
nonceBytes[:])
                r := new(big.Int).SetBytes(hash[:])
                return r.Div(math.MaxBig256, r)
        }
}
```

Recommendation

Add a check to avoid dividing by zero.

Status

Acknowledged.

VCT-004

A zero-gas transaction would panic the chain

Status
Solved

Resolution
Acknowledged

Risk
None

Impact
Recommendation

Likelihood
-

Location

`thor/thorclient/thorclient.go`

Description

An attacker that can bypass other validations and sneak-in a zero-gas transaction so that it reaches the `OverallGasPrice` message can panic a node due a division by zero. Note that `OverallGasPrice` is called not only from consensus-related code but also from the mempool, where validations are more lax. While Coinspect found no way to practically exploit this due to previous checks in the flow, adding a check for zero before dividing is recommended to improve defense in depth.

The root cause can be seen in `OverallGasPrice` in the line just before the `return`:

```
// OverallGasPrice calculate overall gas price.
// overallGasPrice = gasPrice + baseGasPrice * wgas/gas.
func (t *Transaction) OverallGasPrice(baseGasPrice *big.Int, provedWork
*big.Int) *big.Int {
        ...
        x := new(big.Int).SetUint64(wgas)
        x.Mul(x, baseGasPrice)
        x.Div(x, new(big.Int).SetUint64(t.body.gas()))
        return x.Add(x, gasPrice)
}
```

Recommendation

Add a check to avoid dividing by zero.

Status

Acknowledged.

VCT-005

A well-positioned attacker can manipulate `COM()` vote of block

Status
Solved

✓

Resolution
Acknowledged

Risk
None

Impact
Recommendation

Likelihood
-

Location

`thor/thorclient/thorclient.go`

Description

An attacker can listen for a new block from a valid block producer, freely change its `COM()` boolean, and broadcast the new block over the network. A receiver of the block can either see first the original or the modified block. Receivers will ignore the second block received.

The root cause of the issue is that the `signingFields` of the header do not include the `extension` field in pre-`Galactica` blocks:

```go
func (h *Header) signingFields() []any {
        fields := []any{
                h.body.ParentID,
                h.body.Timestamp,
                h.body.GasLimit,
                h.body.Beneficiary,
                h.body.GasUsed,
                h.body.TotalScore,

                &h.body.TxsRootFeatures,
                h.body.StateRoot,
                h.body.ReceiptsRoot,
        }
        if h.body.Extension.BaseFee != nil {
                fields = append(fields, &h.body.Extension)
        }
        return fields
}
```

If an attacker were to exploit this, it would impact the justification process of the 1-bit consensus mechanism. This issue is informational only because the `Galactica` fork already contains a fix for the problem: if the `BaseFee` of a block is not null (a requirement for Galactica blocks), the whole extension field is signed:

```
if h.body.Extension.BaseFee != nil {
    fields = append(fields, &h.body.Extension)
}
```

An attacker can also exploit the same issue with regards to the `Alpha` slice of the `extension` struct.

Recommendation

Consider scanning the blockchain for this particular scenario to see if it has already been exploited.

Consider a hot-fix to add signing to the `extension` field independently of the `Galactica` update.

Status

Acknowledged. This issue will be non-exploitable as soon as the Galactica update is live.

6. Disclaimer

The contents of this report are provided "as is" without warranty of any kind. Coinspect is not responsible for any consequences of using the information contained herein.

This report represents a point-in-time and time-boxed evaluation conducted within a specific timeframe and scope agreed upon with the client. The assessment's findings and recommendations are based on the information, source code, and systems access provided by the client during the review period.

The assessment's findings should not be considered an exhaustive list of all potential security issues. This report does not cover out-of-scope components that may interact with the analyzed system, nor does it assess the operational security of the organization that developed and deployed the system.

This report does not imply ongoing security monitoring or guaranteeing the current security status of the assessed system. Due to the dynamic nature of information security threats, new vulnerabilities may emerge after the assessment period.

This report should not be considered an endorsement or disapproval of any project or team. It does not provide investment advice and should not be used to make investment decisions.