VeChain - VeChainThor Hayabusa Upgrade

Security Assessment

October 17, 2025

Prepared for:
Neil Brett
VeChain

Prepared by: **Anish Naik and Guillermo Larregay**

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Tara Goodwin-Ruffus, Project Manager
tara.goodwin-ruffus@trailofbits.com

The following engineering director was associated with this project:

Benjamin Samuels, Engineering Director, Blockchain
benjamin.samuels@trailofbits.com

The following consultants were associated with this project:

Anish Naik, Consultant
anish.naik@trailofbits.com

Guillermo Larregay, Consultant
guillermo.larregay@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
September 12, 2025	Pre-project kickoff call
September 23, 2025	Status update meeting #1
September 30, 2025	Status update meeting #2
October 6, 2025	Delivery of report draft and report readout meeting
October 9, 2025	Delivery of report draft with fix review addendum
October 17, 2025	Delivery of final report with fix review addendum

Executive Summary

Engagement Overview

VeChain engaged Trail of Bits to review the security of VeChainThor, an EVM-compatible, layer 1 blockchain that is going through a planned hardfork, Hayabusa. This fork will migrate the consensus mechanism of the chain from proof of authority (PoA) to delegated proof of stake (DPoS). The focus of the audit was a differential review of the Hayabusa hardfork; the `release/hayabusa` branch at `180c5ba` was compared to the `master` branch at `8e7b0d3`. Note that the VeChain team independently identified an issue during the review, which was subsequently fixed at `261fabe`.

A team of two consultants conducted the review from September 15 to October 3, 2025, for a total of six engineer-weeks of effort. Our testing efforts focused on identifying issues related to the internal bookkeeping and arithmetic of staking and delegations that could lead to a theft of funds or a chain halt; issues related to incorrect proposer selection and scheduling; and issues related to incorrect business logic for checkpoint justification and finalization. With full access to source code and documentation, we performed static and dynamic testing of the system, using automated and manual processes.

Observations and Impact

Outside of the discovery of TOB-VECHAIN-THOR-1, which could cause a complete denial of service (DoS), we did not identify any other issues that could adversely affect the confidentiality, integrity, or availability of the system. The logic under review had sufficient data validation, access controls, and arithmetic considerations to prevent malicious validators, delegators, or third parties from stealing funds or triggering a chain split or chain halt. Additionally, the system is well-tested using unit tests and E2E tests. Our primary observation is that the system uses smart contract balances to assert on any critical system invariants; we recommend avoiding this, and opting instead to track balances using separate bookkeeping variables that cannot be manipulated by using airdrops, `SELFDESTRUCT`, or direct transfers.

The other primary observation is marked in TOB-VECHAIN-THOR-2, which discusses the risk of using DPoS as a consensus mechanism. Empirical evidence shows that DPoS systems can lead to a plutocracy where a small number of validators hold a large majority of the total stake locked in the blockchain. As the VeChain team continues to progress towards more decentralization, we recommend that the VeChain team consider other variants of PoS that are more resilient to cartel formations, such as NPoS or pure PoS.

Recommendations

Based on the codebase maturity evaluation findings identified during the security review, Trail of Bits recommends that VeChain take the following steps prior to launching the Hayabusa hardfork:

- **Remediate the findings disclosed in this report.** These findings should be addressed through direct fixes or broader refactoring efforts.
- **Develop a fuzzing harness to validate critical arithmetic invariants.** The system's solvency and robustness rely on many arithmetic invariants that must be upheld across all delegator/validator operations and during epoch transitions. Developing and maintaining a fuzzing harness that automatically performs an arbitrary sequence of actions and then asserts on the arithmetic invariants can be used to deepen the confidence in the system's internal bookkeeping. This harness can be integrated into the CI pipeline to monitor whether future code changes violate these invariants.
- **Increase cooldown period and staking cycles lengths.** A short cooldown period and short staking cycles increases validator churn and reduces long-term economic commitment to the ecosystem. In general, the validators should experience friction when trying to exit and re-enter the system. Additionally, consider migrating to other PoS algorithms, such as NPoS or pure PoS, as the system progresses towards more decentralization.

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	2
Medium	0
Low	0
Informational	1
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Configuration	2
Data Validation	1

Project Goals

The engagement was scoped to provide a security assessment of VeChainThor. Specifically, we sought to answer the following non-exhaustive list of questions:

- Is the Hayabusa consensus upgrading fork implemented according to the VIP-253 and VIP-254 specifications?
- Can existing validators migrate correctly during the transition period from PoA to DPoS? Does the transition period take into account the progressive stake migration of validators and delegators?
- Are delegators and validators correctly rewarded? Is it possible for a malicious actor to receive incorrect rewards or game the reward calculations?
- Is the arithmetic and internal bookkeeping logic vulnerable to overflows, precision loss, or division-by-zero issues that could lead to fund theft?
- Does the BFT protocol correctly update the justified/committed checkpoint and checkpoint quality?
- Does the proposer selection algorithm use an unpredictable, deterministic seed? Is the random, weighted selection algorithm implemented correctly?
- Are asset-managing smart contracts implementing safeguards to prevent fund theft or locking? Are access controls implemented correctly for privileged operations?
- Can an unprivileged actor crash or otherwise disrupt the network during the transition period or after it is completed? Can malicious privileged actors do the same?

Project Targets

The engagement involved a review and testing of the following target.

vechain

Repository	https://github.com/vechain/thor
Version	`180c5ba3553f41b0b815079cd818bb9292e86767`
Type	Solidity, Golang
Platform	VeChainThor

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- **Staking and delegations.** We comprehensively reviewed the `Staker` contract and associated logic in the `builtin/staker`, `builtin/solidity`, and `builtin/gascharger` packages. We manually reviewed all validator and delegation operations that are initiated from the `Staker` contract and investigated the following:
    - We reviewed the arithmetic associated with all the operations in the native staker contract (e.g., adding a validator, increasing validator stake, adding a delegator) for potential overflows (that could lead to a theft of funds), precision loss, and incorrect conversions to and from VET and wei.
    - We reviewed all operations in the native staker contract (e.g., adding a validator, increasing validator stake, adding a delegator) for general correctness and to determine if the necessary state changes are made to prevent undefined or malicious behavior.
    - We reviewed the `Staker` contract for access control bypasses and general data validation concerns.
    - We reviewed the Solidity convenience package for nil pointer dereferences, incorrect gas accounting, key collisions, and error handling.
- **BFT protocol updates.** We reviewed the changes made to the BFT protocol. Specifically, we confirmed that the logic around how a checkpoint is justified and committed has been updated. We performed a manual review of this logic to assess whether the COM and non-COM votes are correctly accounted for based on the validator's weight and whether the justified and committed checkpoints are calculated correctly.
- **Proposer selection, block proposals, and block validation.** The proposer selection and schedule was updated to select the next proposer based on their weight. Updates were also made to the block proposal and block validation paths to support the transition to DPoS and the new VTHO reward mechanism. We performed a manual review of this logic for general correctness and to determine if the same seed can be deterministically derived by all validators, if validators are marked offline during inactivity, and if the next proposer is selected based on their proportional weight.
- **Housekeeping and epoch transitions.** We holistically reviewed the DPoS transition and housekeeping logic, which is responsible for activating the DPoS consensus

mechanism and migrating stakes between epochs, evicting validators, exiting validators, and activating new validators, respectively. We performed a manual review of this logic and investigated the following:

  - We reviewed the contract balance check invariant to identify whether a malicious validator, delegator, or third party could adversely affect the invariant to cause a chain halt. This led to the discovery of TOB-VECHAIN-THOR-1, which highlights that an attacker could use SELFDESTRUCT to artificially inflate the `Staker` contract's balance and cause the invariant to break.
  - We reviewed the arithmetic and internal bookkeeping for arithmetic overflows, inconsistencies, or precision loss that could lead to a theft of funds or a chain halt.
  - We reviewed the validator eviction, exiting, and activation logic for general correctness.
  - We reviewed whether there is any operation that can be executed by a validator, delegator, or third party that could adversely affect the DPoS transition or housekeeping logic to cause undefined behavior or a chain halt.

- **VTHO reward distributions.** We manually reviewed the changes made to VTHO reward distribution mechanism (as per VIP-254) as follows:
  - We validated whether VTHO rewards are halted and calculated correctly after the growth stop block number is reached.
  - We reviewed the reward calculation arithmetic for general correctness and to assess whether the proposer and bonded delegators get the correct proportions of the rewards.
  - We reviewed the block proposal and validation paths to determine if the rewards are distributed to the proposer and bonded delegators.
  - We reviewed the storage writes for delegation rewards for key collisions.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- **Third-party dependencies.** We did not review any third-party dependencies that the VeChainThor client may depend on.

- **StarGate:** StarGate is the trusted delegation smart contract that all delegators must interact with to delegate to a validator. This smart contract was considered out of scope. Thus, we assume that the smart contract has sufficient access controls to prevent a malicious actor from impersonating an honest delegator.
- **Slashable behavior.** Slashing is currently not implemented. Thus, we did not evaluate the system's ability to handle "slashable behavior" such as double-signing attacks.
- **De-prioritized packages.** The diff that we reviewed has a number of files/packages that we de-prioritized and considered out of scope for this review:
    - `test/`
    - `cmd/`
    - `chain/`
    - `poa/`
    - `thorclient/`

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	Most arithmetic operations use overflow checks. Additionally, the stake for a validator (including bonded delegations) is bounded between a minimum and maximum amount. The arithmetic is sufficiently tested. Additionally, we did not identify any instances of precision loss or division by zero. We recommend developing an E2E fuzzing harness that can sufficiently model all basic operations and evaluate the most critical arithmetic invariants.	**Satisfactory**
Auditing	All smart contract operations emit events. There is sufficient logging to alert on the contract balance check invariant failure, and some other errors are logged downstream. The VeChain team also maintains an off-chain monitoring solution that uses the logs, events, and the client API to track critical system metrics and automatically trigger an alert on Slack in case of an issue. Additionally, the VeChain team is currently building an incident response (IR) plan. More information on developing an IR plan can be found in appendix D.	**Satisfactory**
Authentication / Access Controls	Access controls are implemented for privileged functions, and the executor implements a two-step process. We did not identify any opportunities to bypass any access controls or perform an unauthorized action.	**Strong**
Complexity Management	The system is generally well-composed and easy to reason through. PoA- and DPoS-related logic are logically separated, which allowed us to easily assess the behavior	**Satisfactory**

	of the system between the two consensus modes.	
Cryptography and Key Management	No cryptographic primitives (except for generating the seed for proposer selection) were considered during this audit.	**Not Applicable**
Decentralization	As highlighted in TOB-VECHAIN-THOR-2, DPoS systems are vulnerable to the formation of a plutocracy that adversely affects the decentralization of the system.	**Moderate**
Documentation	The VIP-253 and VIP-254 specifications are accurate and provide sufficient context to review the system. Additionally, the code has sufficient inline comments to reason about the behavior of various functions. We had access to non-public, internal documentation during the audit, which aided in understanding the system at a high level. The publicly available VeChain documentation on the website is correctly divided into end users and developers. We recommend making all documentation public once the Hayabusa fork is live.	**Satisfactory**
Low-Level Manipulation	There was no usage of low-level assembly in scope.	**Not Applicable**
Testing and Verification	The system is well-tested using unit tests and E2E tests. We recommend developing a fuzzing harness to dynamically test the critical arithmetic invariants.	**Satisfactory**
Transaction Ordering	We did not identify any transaction-ordering risks present within the system. The only viable vector is if a validator front-runs another validator to enter the FIFO queue.	**Not Applicable**

Summary of Findings

The table below summarizes the findings of the review, including details on type and severity.

ID	Title	Type	Severity
1	Staker contract balance invariant can be broken using SELFDESTRUCT	Data Validation	**High**
2	DPoS is vulnerable to significant centralization risk	Configuration	**High**
3	Short cooldown and short staking cycles increase validator churn	Configuration	**Informational**

Detailed Findings

1. Staker contract balance invariant can be broken using SELFDESTRUCT

Severity: **High**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-VECHAIN-THOR-1
Target: `builtin/staker/housekeep.go`	

Description

VeChain DPoS uses a `Staker` contract to handle validations and delegations. One of the system's invariants requires that the total amount of staked VET matches the `Staker` contract balance.

Even though the contract prevents VET from being received via normal transfers, an attacker can still send VET to the contract via SELFDESTRUCT. This can break two assumptions: the first one is that the amount of VET transferred to the contract is always an integer, and the second is that all VET in the contract comes from a stake or delegation operation.

```
func (s *Staker) ContractBalanceCheck(pendingWithdraw uint64) error {
	[...]
	// Get the staker contract's account balance
	stakerAddr := s.Address()
	balance, err := s.state.GetBalance(stakerAddr)
	if err != nil {
		return err
	}
	balanceVET := ToVET(balance)
	total := uint64(0)
	for _, stake := range []uint64{lockedStake, queuedStake, withdrawableStake,
cooldownStake, pendingWithdraw} {
		partialSum, overflow := math.SafeAdd(total, stake)
		if overflow {
			return fmt.Errorf(
				"total overflow occurred while adding locked(%d) +
queued(%d) + withdrawable(%d) + cooldown(%d) + pendingWithdraw(%d)",
				lockedStake,
				queuedStake,
				withdrawableStake,
				cooldownStake,
				pendingWithdraw,
			)
		}
```

```
                total = partialSum
        }
        if balanceVET != total {
                logger.Error("sanity check failed",
                        "locked", lockedStake,
                        "queued", queuedStake,
                        "withdrawable", withdrawableStake,
                        "cooldown", cooldownStake,
                        "pendingWithdraw", pendingWithdraw,
                        "total", total,
                        "balance", balanceVET,
                )
                return fmt.Errorf(
                        "sanity check failed: locked(%d) + queued(%d) + withdrawable(%d)
= %d, but account balance is %d, diff= %d",
                        lockedStake,
                        queuedStake,
                        withdrawableStake,
                        total,
                        balanceVET,
                        balanceVET-total,
                )
        }
        return nil
}
```

Figure 1.1: An excerpt of the `ContractBalanceCheck` function (thor/builtin/staker/housekeep.go#L262–L324)

As shown in figure 1.1, the `ContractBalanceCheck` invariant can be broken by forcing VET into the contract. Since the invariant is checked on housekeeping, staking operations, and block validation, the attacker can do the following:

- Prevent the PoA to DPoS transition from completing if the balance is altered during the transition period
- Prevent block production if a transaction is sent after DPoS is activated, since blocks containing transactions that break the balance invariant are invalid

Exploit Scenario

Mallory develops a contract that, when deployed, calls the SELFDESTRUCT opcode and sends its balance to the staker contract. Then she deploys and funds it with more than 1 VET, altering the balance of the staker contract by enough to break the invariant.

Recommendations

Short term, track the `Staker` contract's VET balance using an additional storage variable that is updated only on direct calls to the smart contract. Direct transfers and airdrops should not affect this variable.

Long term, do not rely on contract balances, as they can be altered. Improve the test suite to consider adversarial cases. Additionally, consider reducing the reliance on the contract balance check on critical code paths (e.g., block validation), since this opens up a significant DoS attack vector.

2. DPoS is vulnerable to significant centralization risk	
Severity: **High**	Difficulty: **N/A**
Type: Configuration	Finding ID: TOB-VECHAIN-THOR-2
Target: N/A	

Description

DPoS systems like VeChain are susceptible to centralization risks due to the formation of a plutocracy that governs a large majority of the total stake and voting power.

More specifically, there is historical precedence of DPoS systems being vulnerable to vote buying and bribery attacks. In VeChain, the validators that have the highest total stakes have the highest likelihood of proposing the next block. This will naturally incentivize delegators to bond to those validators since this will increase their likelihood of receiving VTHO rewards. More importantly, larger validators can also offer kickbacks or bribes to delegators (outside of the 70% delegation reward) to further incentivize delegators to bond to them.

Over time, this will create a concentrated group of validators that reach the maximum stake possible of 600M VET. At the same time, there will be a group of validators with insufficient stake that are unlikely to propose blocks and will choose to exit the network.

As soon as these validators exit the network, the validators with the largest stakes are economically incentivized to spin up new validators and participate in the protocol. The FIFO activation queue is ineffective against a Sybil attack if the total stake gets concentrated into the hands of a few validators and pushes other, smaller validators to exit the network.

It is important to note that under PoA, there is no economic incentive to act maliciously and form a plutocracy of the largest stakeholders. With the migration to DPoS, the original PoA validator set is economically incentivized to push out other validators by accruing as much stake as possible and then activating new validators under different identities to capture an even larger portion of the total stake.

Recommendations

Long term, consider migrating to a variant of PoS, such as nominated PoS (NPoS) or a pure PoS mechanism like Ethereum. It is important to note that there are no production-grade DPoS systems that have implemented any mitigations that would prevent cartel formation and the concentration of stake.

References

- A Node Operator's Confession of EOS DPoS Corruption
- Governance, Part 2: Plutocracy Is Still Bad

3. Short cooldown and short staking cycles increase validator churn

Severity: **Informational**	Difficulty: **N/A**
Type: Configuration	Finding ID: TOB-VECHAIN-THOR-3
Target: N/A	

Description

A 24-hour cooldown period and short staking cycles (e.g., seven days) make joining/exiting timing highly predictable and enable low-friction validator churn that undermines liveness and economic stickiness.

With a 24-hour cooldown, an operator can exit and be eligible to re-enter after roughly 48 epochs (each epoch is 30 minutes). Because epochs are short and exits are processed at epoch boundaries, exits can be precisely timed with many daily opportunities. Even with a one-exit-per-epoch cap, the network can process a non-trivial number of exits per day. If activation throughput per epoch does not reliably match or exceed exits, the active set can linger below target across multiple epochs, increasing missed slots and elongating time-to-finality. While this does not threaten consensus safety, it degrades performance and weakens deterrence against poor uptime, since the cost of temporarily leaving is low.

Short staking cycles reduce long-term economic commitment and encourage opportunistic participation. Operators can stake for a brief window to capture rewards and exit before adverse conditions or penalties materialize, re-queuing shortly after the short cooldown elapses. The frequent epoch cadence makes this behavior easier to script and synchronize, and delegation flows may follow, amplifying weight volatility around epoch boundaries. Over time, this contributes to sustained, rate-limited churn that erodes long-term commitment incentives and increases the system's reliance on continuous backfilling to meet liveness targets.

Recommendations

Short term, lengthen the cooldown period to a larger number of days (if not weeks) to disincentivize validators from exiting and re-entering the system after a short period of time. Additionally, lengthen the minimum, medium, and long-term staking periods to incentivize long-term economic commitment to the security of the chain.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage
Transaction Ordering	The system's resistance to transaction-ordering attacks

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category does not apply to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Code Quality Recommendations

This appendix contains recommendations that do not have immediate or obvious security implications. However, implementing them may enhance the code's readability and may prevent the introduction of vulnerabilities in the future.

- **Remove the inline comment on line 208 in figure C.1.** The `StopEnergyGrowth` function does not throw an error if it is called more than once. The inline comment is inaccurate.

```
 207    // StopEnergyGrowth sets the end time of energy growth at the current
block time.
 208    // Calling this function more than once will result an error.
 209    func (e *Energy) StopEnergyGrowth() error {
```

Figure C.1: An excerpt of the `StopEnergyGrowth` function (thor/builtin/energy/energy.go#L207–L209)

- **Solve all remaining TODOs in comments.** Issues and TODOs should be tracked in an issue tracker instead of using in-code comments. Some examples are shown below:

```
// Current Status:
//   - Covers basic read operations (no arguments)
//   - TODO: Implement argument handling for functions that require parameters
//   - TODO: Add test cases for state-modifying operations
```

Figure C.2: Pending TODOs in tests (thor/builtin/staker_native_gas_test.go#L15–L18)

```
},
// TODO: How can we mint thousands of blocks and perform housekeeping?
{
```

Figure C.3: Pending TODOs in tests (thor/builtin/staker_native_gas_test.go#L203–L205)

D. Incident Response Recommendations

This section provides recommendations on formulating an incident response plan.

- **Identify the parties (either specific people or roles) responsible for implementing the mitigations when an issue occurs (e.g., deploying smart contracts, pausing contracts, upgrading the front end, etc.).**
- **Document internal processes for addressing situations in which a deployed remedy does not work or introduces a new bug.**
    - Consider documenting a plan of action for handling failed remediations.
- **Clearly describe the intended contract deployment process.**
- **Outline the circumstances under which VeChain will compensate users affected by an issue (if any).**
    - Issues that warrant compensation could include an individual or aggregate loss or a loss resulting from user error, a contract flaw, or a third-party contract flaw.
- **Document how the team plans to stay up to date on new issues that could affect the system; awareness of such issues will inform future development work and help the team secure the deployment toolchain and the external on-chain and off-chain services that the system relies on.**
    - Identify sources of vulnerability news for each language and component used in the system, and subscribe to updates from each source. Consider creating a private Discord channel in which a bot will post the latest vulnerability news; this will provide the team with a way to track all updates in one place. Lastly, consider assigning certain team members to track news about vulnerabilities in specific system components.
- **Determine when the team will seek assistance from external parties (e.g., auditors, affected users, other protocol developers) and how it will onboard them.**
    - Effective remediation of certain issues may require collaboration with external parties.
- **Define contract behavior that would be considered abnormal by off-chain monitoring solutions.**

It is best practice to perform periodic dry runs of scenarios outlined in the incident response plan to find omissions and opportunities for improvement and to develop “muscle memory.” Additionally, document the frequency with which the team should perform dry runs of various scenarios, and perform dry runs of more likely scenarios more regularly. Create a template to be filled out with descriptions of any necessary improvements after each dry run.

E. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On October 9, 2025, Trail of Bits reviewed the fixes and mitigations implemented by the VeChain team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the three issues described in this report, VeChain has resolved TOB-VECHAIN-THOR-1 and has not resolved TOB-VECHAIN-THOR-3. We are marking TOB-VECHAIN-THOR-2 with an Undetermined fix status since the issue requires a strategic, long-term initiative to migrate from DPoS and cannot be evaluated during the course of a fix review.

For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Staker contract balance invariant can be broken using SELFDESTRUCT	Resolved
2	DPoS is vulnerable to significant centralization risk	Undetermined
3	Short cooldown and short staking cycles increase validator churn	Unresolved

Detailed Fix Review Results

TOB-VECHAIN-THOR-1: Staker contract balance invariant can be broken using SELFDESTRUCT

Resolved in PR#1394. The VeChain team has updated the `Staker` contract to now track the total VET deposited in the system using a separate state variable. This state variable is updated only when funds are deposited or withdrawn from the system by validators or delegators. Thus, this variable cannot be manipulated by direct token transfers or SELFDESTRUCT operations. The contract balance invariant check has also been updated to validate that the actual VET balance of the contract is greater than the value stored in the state variable. Since the check is no longer a strict equality, the SELFDESTRUCT operation would only benefit the system and not cause a DoS.

TOB-VECHAIN-THOR-2: DPoS is vulnerable to significant centralization risk
Since the fix for this issue is a larger, long-term initiative and cannot be evaluated during a fix review, we are marking the resolution as Undetermined.

The VeChain team has provided the following context for this finding's fix status:

> *VeChain recognizes that Delegated Proof of Stake (DPoS) systems may face theoretical risks of stake concentration. VeChain's implementation combines transparent on-chain governance, community participation, stake-cap limits, and a validator activation queue to discourage collusion or dominance. Future protocol revisions may consider adjustments to staking and delegation parameters to maintain a decentralized, secure, and resilient network.*

TOB-VECHAIN-THOR-3: Short cooldown and short staking cycles increase validator churn
Unresolved. The VeChain team has provided the following context for this finding's fix status:

> *VeChain acknowledges that shorter cooldowns and staking cycles can, in theory, contribute to increased validator turnover. VeChain will continue to assess validator participation dynamics and staking behaviours to ensure optimal network stability, liveness, and fairness. Future protocol revisions may consider adjustments to staking parameters or cooldown durations as part of the ongoing effort to enhance validator commitment and overall network resilience.*

F. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review assessments, supporting client organizations in the technology, defense, blockchain, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, Uniswap, Solana, Ethereum Foundation, Linux Foundation, and Zoom.

To keep up with our latest news and announcements, please follow @trailofbits on X or LinkedIn and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

Trail of Bits performed all activities associated with this project in accordance with a statement of work and an agreed-upon project plan.

Security assessment projects are time-boxed and often rely on information provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test software controls and security properties. These techniques augment our manual security review work, but each has its limitations. For example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. A project's time and resource constraints also limit their use.