BeethovenX Sonic Staking

Security Assessment

January 10, 2025

Prepared for:
Daniel
BeethovenX DAO, LLC

Prepared by: **Bo Henderson**

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Mary O'Brien, Project Manager
mary.obrien@trailofbits.com

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultant was associated with this project:

Bo Henderson, Consultant
bo.henderson@trailofbits.com

Project Timeline

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

Date	Event
January 6, 2025	Project kickoff
January 10, 2025	Delivery of report draft
January 13, 2025	Report readout meeting
January XX, 2025	Delivery of final comprehensive report

Executive Summary

Engagement Overview

BeethovenX engaged Trail of Bits to review the security of its Sonic Staking application. The `SonicStaking` contract provides a liquid staking token (LST) that allows users to deposit Sonic tokens, mint staked Sonic shares, and earn yield by delegating funds to validators.

One consultant conducted the review from January 6 to January 10, 2025, for a total of one engineer-week of effort. Our testing efforts focused on verifying the safety of user funds and the correctness of fixes implemented in response to a previous security audit. With full access to source code and documentation, we performed static and dynamic testing of the Sonic Staking repository, using automated and manual processes.

Observations and Impact

Our security assessment of the BeethovenX Sonic Staking application did not uncover any issues that warrant an immediate upgrade of the deployed `SonicStake` contract.

One low-severity issue was found in an error-prone feature of the pause process (TOB-BEETS-1). However, we recommend focusing on educating contract administrators about this issue instead of directly remediating it.

The codebase is well organized, and test coverage is extremely thorough.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that BeethovenX take the following steps:

- **Remediate the findings disclosed in this report during the next upgrade.** If an upgrade is conducted in the future, fixes for the reported issues should be included.
- **Expand the public documentation.** Although the current Beets documentation provides a high-level overview of the Beets ecosystem's services, usage instructions for the Sonic Staking system are insufficient. In particular, the risks of slashing should be included in the FAQ at the bottom of the Stake page.
- **Create an incident response plan**. See appendix E for guidance.

Finding Severities and Categories

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

EXPOSURE ANALYSIS

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

CATEGORY BREAKDOWN

Category	*Count*
Data Exposure	1
Timing	2

Project Targets

The engagement involved reviewing and testing the following target.

Sonic Staking

Repository	https://github.com/beethovenxfi/sonic-staking
Version	`4fde3014f287f2ba38d559a23317a892e92cd3e6`
Type	Solidity
Platform	Sonic

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:

- A manual review of the `SonicStaking` and `DeploySonicStaking` smart contracts as well as all associated documentation
- A survey of the `SonicStaking` contract deployed to the Sonic blockchain at address `0xE5DA20F15420aD15DE0fa650600aFc998bbE3955`
    - We reviewed the transactions used to deploy and initialize the system as well as the state variables of this and supporting contracts, such as privileged multisignature wallets and the SFC contract, in an effort to verify that the target contract is interacting with them safely and that configuration values such as timeouts are aligned.
- A group walkthrough of the `SonicStaking` contract with five other Trail of Bits teammates to generate leads for potential security issues
- A manual review of all fixes applied in response to issues found during the previous security audit
- Static analysis of the source code using Slither
    - We triaged all findings and found no noteworthy issues.
- Dynamic analysis of the unit test coverage using `slither-mutate` and a manual review of the results, finding no gaps in test coverage

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:

- Although we surveyed supporting components such as multisignature wallets and the SFC contract, the source code of these contracts was not a primary focus of this review. Bugs that impact the security of the Sonic Staking system may be present in these components.

Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in-house, to perform automated testing of source code and compiled software.

Test Harness Configuration

We used the following tools in the automated testing phase of this project:

Tool	Description	Policy
Slither	A static analysis framework that can statically verify algebraic relationships between Solidity variables	Appendix D
`slither-mutate`	A deterministic mutation generator that detects gaps in test coverage	Appendix D
`zizmor`	A static analysis tool that can find many common security issues in typical GitHub Actions CI/CD setups	Appendix D

Areas of Focus

Our automated testing and verification work focused on the following:

- Identify common issues and anti-patterns in Solidity and Python
- Identify gaps in test coverage to guide our review efforts

Test Results

The results of this focused testing are detailed below.

`slither-mutate`: The following table displays the portion of each mutant type for which all unit tests passed. The presence of uncaught mutants indicates gaps in test coverage because the test suite did not catch the introduced change.

- Uncaught revert mutants replace a given expression with a `revert` statement, indicating that the line is not executed during testing.
- Uncaught comment mutants comment out a given expression and indicate that the effects of this line are not checked by any assertions.
- Uncaught tweak mutants indicate that the expression being executed features edge cases not covered by the test suite.

The `sonic-staking/src` subdirectory is the root for all target paths listed below. Contracts that support tests or that produced zero analyzed mutants (e.g., interfaces) were omitted from mutation testing analysis.

Target	Uncaught Reverts	Uncaught Comments	Uncaught Tweaks
`SonicStaking.sol`	0%	0%	0%

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	The `SonicStaking` contract uses a modern version of Solidity without any unchecked arithmetic blocks, preventing uncaught overflows. Arithmetic that could revert due to an overflow is clearly commented. Although precision-losing operations round consistently, rounding direction is not explicitly in favor of the protocol. However, the minimum deposit and undelegate amounts prevent rounding errors from meaningfully impacting the pool's solvency.	**Satisfactory**
Auditing	The system consistently emits events that include enough data for effective off-chain monitoring. Event and parameter names provide helpful context; however, the system would benefit from an incident response plan. Although automated agents provide implicit off-chain monitoring, this component was not in scope, and the system may benefit from a more explicit monitoring solution.	**Satisfactory**
Authentication / Access Controls	Privileges are divided among granular roles, which rigorously follow the principle of least privilege. We did not identify any missing or misconfigured access controls.	**Strong**
Complexity Management	The `SonicStaking` contract is composed of one Solidity file that is divided into sections such as end-user, admin, and operator-related functions. The business logic as a whole is optimized for readability. All functions have a well-defined responsibility, and none feature high cyclomatic complexity or overly nested operations.	**Strong**

Cryptography and Key Management	Although the `SonicStaking` contract initializes ownership to `msg.sender`, which is exposed to the development environment, the factory contract transfers ownership to a time-locked multisignature wallet as part of the deployment process.	**Satisfactory**
Decentralization	Privileged roles cannot unilaterally withdraw user funds, and the project's README clearly enumerates and describes these roles. All system logic is upgradeable by a time-locked multisignature wallet. The time lock for upgrading (three weeks) is longer than the withdrawal delay (two weeks), giving users a reliable opportunity to opt out of undesired upgrades.	**Satisfactory**
Documentation	Each function has NatSpec comments that describe the function's purpose and all parameters. However, these comments would benefit from additional descriptions of return values and revert conditions. The repository README provides a clear description of technical details and a discussion of design decisions. Public, high-level documentation is available. However, additional usage instructions specific to the Sonic Staking application are required for users to interact safely with the service.	**Moderate**
Low-Level Manipulation	This system does not use assembly except via well-established libraries. Low-level calls are used only to transfer native tokens in alignment with recommended practices.	**Not Applicable**
Testing and Verification	The test suite is thorough, featuring both unit and fuzz tests. These tests cover more than 99.5% of statements, and mutation testing did not uncover any gaps in test coverage.	**Strong**
Transaction Ordering	Such staking systems are inherently sensitive to timing risks, especially surrounding the claiming of rewards and the realization of losses due to validator slashing. Although these transaction ordering risks are well managed in general, the pause process is at risk of failing due to out-of-order transactions (TOB-BEETS-1).	**Satisfactory**

Summary of Findings

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

ID	Title	Type	Severity
1	Global pause method will fail if a subcomponent is already paused	Timing	**Low**
2	Risk of read-only reentrancy via Balancer pools	Timing	**Informational**
3	Credentials are persisted in the GitHub CI workflow	Data Exposure	**Informational**

Detailed Findings

1. Global pause method will fail if a subcomponent is already paused

Severity: **Low**	Difficulty: **High**
Type: Timing	Finding ID: TOB-BEETS-1
Target: `src/SonicStaking.sol`	

Description

Internal pause methods, such as `_setUndelegatePaused`, will revert while attempting to set a pause flag to the current value (figure 1.1). This could cause unintended reversions if the global pause method is called after a more specific feature has already been paused (figure 1.2).

```
function _setUndelegatePaused(bool newValue) internal {
    require(undelegatePaused != newValue, PausedValueDidNotChange());

    undelegatePaused = newValue;
    emit UndelegatePausedUpdated(msg.sender, newValue);
}
```

Figure 1.1: An internal, feature-specific pause method at SonicStaking.sol#L818–L823

```
function pause() external onlyRole(OPERATOR_ROLE) {
    _setDepositPaused(true);
    _setUndelegatePaused(true);
    _setUndelegateFromPoolPaused(true);
    _setWithdrawPaused(true);
}
```

Figure 1.2: The external, global `pause` method at SonicStaking.sol#L571–L576

Exploit Scenario

Abnormal behavior related to undelegations is detected on-chain, so the admin calls `setUndelegatePaused` to halt this feature. Later, a serious bug is discovered that affects user withdrawals. The operator calls the `pause` function to halt all contract features, but the call fails because one protocol feature is already paused. As a result, the team's response to this incident is delayed.

Recommendations

Short term, ensure the administration team is aware of this issue and document it clearly in internal runbooks. Keep this finding in mind as you review appendix E, regarding incident response.

Long term, move the `PausedValueDidNotChange` requirements into external feature-specific pause methods such as `setUndelegatePaused` so that the global pause method will always succeed in pausing the system.

2. Risk of read-only reentrancy via Balancer pools

Severity: **Informational**	Difficulty: **High**
Type: Timing	Finding ID: TOB-BEETS-2
Target: `src/SonicStaking.sol`	

Description
The internal `_withdraw` method passes execution control to an arbitrary `user` account while sending native tokens. Although all state-modifying methods of the `SonicStaking` contract feature robust reentrancy protection, the `getRate` method can be called by external Balancer pools, resulting in out-of-order events.

Exploit Scenario
A user withdraws from the `SonicStaking` contract to another smart contract, which calls a Balancer pool. This pool contract calls the `getRate` method and proceeds to emit events that are tracked by off-chain monitoring tools. When the Balancer pool returns, the `Withdrawn` event is emitted. The resulting out-of-order events cause the off-chain monitoring tool to malfunction.

Recommendations
Short term, monitor interactions between the `SonicStaking` contract and Balancer pools. If off-chain components, such as those managing the operator role, react to events from Balancer pools, ensure that they are robust against out-of-order events.

Long term, add a `nonReentrant` modifier to the `getRate` method or have the method emit the `Withdrawn` event before passing execution control to an arbitrary account.

3. Credentials are persisted in the GitHub CI workflow

Severity: **Informational**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-BEETS-3
Target: `.github/workflows/test.yml`	

Description

By default, the `checkout` GitHub workflow action persists a credential to the `.git/config` file to authenticate future Git actions. This credential does not normally need to be persisted and may be leaked if later CI steps are updated to, for example, upload artifacts.

Recommendations

Short term, update the `checkout` action configuration to set the `persist-credentials` option to `false`.

```
name: Foundry project
runs-on: ubuntu-latest
steps:
  - uses: actions/checkout@v4
    with:
      submodules: recursive
      persist-credentials: false
```

Long term, review details regarding the `artipacked` issue in the `zizmor` documentation.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity, 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 exceeded 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

The following recommendations are not associated with any specific vulnerabilities. However, they will enhance code readability and may prevent the introduction of vulnerabilities in the future.

- **Use Solidity time units.** The `withdrawDelay` state variable is initialized to two weeks by multiplying the number of seconds per week by two. Solidity provides a `weeks` unit, which should be used instead to enhance readability.

D. Automated Analysis Tool Configuration

Slither

We used Slither to detect common issues and anti-patterns in the codebase. Although Slither did not discover any severe issues during this review, integrating Slither into the project's testing environment can help find other issues that may be introduced during further development and will help improve the overall quality of the smart contracts' code.

```
slither . --filter-paths='node_modules,src/mock,src/interfaces,lib'
```

Figure D.1: The Slither command used to analyze the `SonicStaking` contract

Integrating `slither-action` into the project's CI pipeline can automate this process.

slither-mutate

Slither version 0.10.2 introduced a built-in Solidity mutation testing tool with first-class support for Hardhat and Foundry projects. We ran a mutation testing campaign against the target smart contract using the commands shown in figure D.2.

```
forge clean
slither-mutate . --test-cmd="forge test" \
  --solc-remap="$(cat remappings.txt | tr '\n' ' ')" \
  --ignore-dirs='test,mutation_campaign,node_modules,mock,interfaces,lib,script' \
  --timeout=180
```

Figure D.2: A command that runs a mutation testing campaign against the `SonicStaking` contract

Guidance on running mutation tests:

- The overall runtime of this campaign is approximately two hours on a consumer-grade laptop.
- Some variance in test runtime was observed, which resulted in some false negatives. Extending the timeout to three minutes resolved this problem.
- Due to historical differences between Slither and solc, the `--solc-remap` argument is space-delimited, and the `--ignore-dirs` argument is comma-delimited.
- The value provided to the `--solc-remap` argument can be generated inline from the `remappings.txt` file provided in the target repo. If the quotes and bash subshell in this argument prove error-prone, this argument can be generated and provided manually.

`slither-mutate` did not identify any gaps in the `SonicStaking` test coverage. However, if any features are changed in a way that warrants updates to the test suite, then `slither-mutate` may provide actionable guidance regarding newly introduced coverage gaps.

zizmor

We used `zimor` to detect common issues and anti-patterns in the codebase's GitHub workflows.

```
zizmor target/.github/workflows/test.yml
```

Figure D.3: The `zizmor` command used to audit the `sonic-staking` repository's CI configuration

See the `zizmor` documentation for more information.

E. Incident Response Recommendations

This section provides recommendations for 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 BeethovenX 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 current 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.

F. 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 January 14, 2025, Trail of Bits reviewed the fixes and mitigations implemented by the BeethovenX 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, the BeethovenX team resolved one and did not resolve the remaining two. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Global pause method will fail if a subcomponent is already paused	Unresolved
2	Risk of read-only reentrancy via Balancer pools	Unresolved
3	Credentials are persisted in the GitHub CI workflow	Resolved

Detailed Fix Review Results

TOB-BEETS-1: Global pause method will fail if a subcomponent is already paused
Unresolved. BeethovenX has provided the following context:

> *We acknowledge the finding and have already made operators of the staking contract aware of the issue. In addition, we'll add information about this risk to our internal documentation, including the emergency response plan.*

TOB-BEETS-2: Risk of read-only reentrancy via Balancer pools
Unresolved. BeethovenX has provided the following context:

> *We acknowledge the finding. We don't have any off-chain monitoring tools or agents in place that use the `getRate` function.*

TOB-BEETS-3: Credentials are persisted in the GitHub CI workflow
Resolved in PR #64. This update sets the `persist-credentials` option to `false` in the `checkout` action's configuration, deactivating the default behavior to persist credentials.

G. 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 projects, supporting client organizations in the technology, defense, and finance industries and government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter 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.
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
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.