Gyroscope

Security Assessment (Summary Report)

September 15, 2022

Prepared for:
Gyroscope

Prepared by:
Alexander Remie
Michael Colburn
Felipe Manzano
Justin Jacob
Robert Schneider

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 80+ 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, as well as 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.
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

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and mutually agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be 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 the controls and security properties of software. 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. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Executive Summary

Engagement Overview

Gyroscope engaged Trail of Bits to review the security of its smart contracts. From January 10 to January 14, 2022, February 7 to February 11, 2022, and March 21 to April 1, 2022, a team of three consultants conducted a security review of the client-provided source code, with seven person-weeks of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

Project Scope

Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We performed automated testing and a manual review of the code, in addition to running system elements.

The scope of the initial one-week security review was limited to the `gyro-token` contracts, part of the math of the `GyroTwoPool` and `GyroThreePool` contracts, and part of the `PrimaryAMMV1` contract.

The scope for the second one-week security review was limited to the `GyroCEMMPool`, `GyroThreePool`, and `PrimaryAMMV1` contracts.

During the final two-week security review, we focused on the `protocol` repository, specifically the oracle contracts and protocol safety checks. We also reviewed the remaining components of the system and changes introduced to the previously reviewed components after the first two audits.

Summary of Findings

The audit uncovered two significant flaws that could impact system confidentiality, integrity, or availability. A summary of the findings and details on notable findings are provided below.

EXPOSURE ANALYSIS

Severity	Count
High	2
Medium	1
Low	5
Informational	4
Undetermined	4

CATEGORY BREAKDOWN

Category	Count
Access Controls	2
Auditing and Logging	1
Data Validation	8
Undefined Behavior	5

Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

- Certain safety checks performed during minting and redeeming operations do not properly check the returned error codes; as a result, minting and redeeming operations could succeed even if the `ReserveSafetyManager` indicates that they are unsafe.
- An edge case in the `GyroTwoPool` arithmetic could allow users to receive free tokens.

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Cara Pearson, Project Manager
cara.pearon@trailofbits.com

The following engineers were associated with this project:

Alexander Remie, Consultant
alexander.remie@trailofbits.com

Michael Colburn, Consultant
michael.colburn@trailofbits.com

Felipe Manzano, Consultant
felipe.manzano@trailofbits.com

Justin Jacob, Consultant
justin.jacob@trailofbits.com

Robert Schneider, Consultant
robert.schneider@trailofbits.com

Project Timeline

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

Date	Event
January 6, 2022	Pre-project kickoff call
January 18, 2022	Delivery of report draft
January 18, 2022	Report readout meeting
February 7, 2022	Pre-project kickoff call
February 14, 2022	Delivery of report draft
February 14, 2022	Report readout meeting
March 17, 2022	Pre-project kickoff call
March 25, 2022	Status update meeting #1
April 1, 2022	Deliver of report draft
April 1, 2022	Report readout meeting
April 20, 2022	Delivery of final report
August 19, 2022	Review of fixes implemented by Gyroscope
September 15, 2022	Delivery of final report with fix log

Project Targets

The engagement involved a review and testing of the targets listed below.

Gyroscope Token

Repository	https://github.com/gyrostable/gyro-token
Version	`a0dd0d7156ee4d11e2ca90cb71ae17866b566c5b` (week 1)
Type	Solidity
Platform	Ethereum

Gyroscope Vaults

Repository	https://github.com/gyrostable/vaults
Versions	`bd001706c9f8b07e0f85c0b1f1d5c33443f47846` (week 1) `73215adce9f96cc8a479321a4e90f176e4f82a19` (week 2)
Type	Solidity
Platform	Ethereum

Gyroscope Protocol

Repository	https://github.com/gyrostable/protocol
Versions	`e2be442731a4723b6cd61cc65f0a36c62e8850ec` (week 1) `55c5f3647e96ba07f32c8c09f9151b82e01c5abe` (week 2) `69ed2ffdf09879203e85bffbd148cabf0cff3acc` (weeks 3 and 4)
Type	Solidity
Platform	Ethereum

Project Coverage

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

- **`gyro-token/`** contains the Gyro token contracts. We used static analysis, a manual review, and Echidna to test the process of upgrading the existing L1 contract to the new one, the inflation arithmetic, the access controls, the validation of the inputs, and the existing on-chain deployment.

- **`vaults/`** contains the `GyroTwoPool`, `GyroThreePool`, and `GyroCEMMPool` implementations, which are based on Balancer. Our scope was limited to specific parts of the contracts. We used static analysis, a manual review, and Echidna to test the arithmetic, the access controls, the input validation, and the parity between the implementation and white paper. We also manually reviewed the differences between the base contracts and Balancer to ensure that the modifications did not introduce vulnerabilities.

- **`protocol/`** contains the AMM and various other core contracts of the system. Our scope in the initial two weeks was limited to the `PrimaryAMMV1` contract. We used static analysis and a manual review to test the access controls, the input validation, and the parity between the implementation and white paper. In the final weeks of the assessment, we reviewed these contracts to ensure that the oracle components interacted with external protocols properly, that protocol-level safety checks were correctly performed, and that no other general security issues were present.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. During this project, we were unable to perform comprehensive testing of the following system element, which may warrant further review:

- End-to-end property-based testing of `protocol/` with Echidna

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 arithmetic performed by the various AMM implementations is, at times, quite complex but is thoroughly specified in white papers. Math libraries are used where appropriate. We identified some precision-related edge cases early in the engagement, but the code was updated with architectural changes and additional tests in later weeks to address them.	**Satisfactory**
Auditing	All functions except for `changeGovernor` in the **gyro-token** repository emit events for important state changes. This makes it easy to monitor the state of the system. We recommend looking into blockchain monitoring tools and developing an incident response plan.	**Moderate**
Authentication / Access Controls	Appropriate access controls are in place for privileged functionality. We recommend always implementing a two-step process for ownership and governor transfers to prevent loss of control of the contracts.	**Satisfactory**
Complexity Management	The codebase is quite complex, and some interactions require tracing through several contracts. Because we reviewed the codebase on a rolling basis as the Gyroscope team finalized certain features, we encountered deprecated contracts and functionality that was not previously implemented.	**Moderate**
Decentralization	The protocol has several privileged actors, such as the governor, oracle guardians, and reserve managers. The	**Satisfactory**

	Gyroscope team has documentation describing some of these roles and the pitfalls of existing governance architectures. We recommend extending this documentation as much as possible and using multisig schemes wherever possible, especially for reserve managers, who have the ability to withdraw funds from the protocol reserves.	
Documentation	Gyroscope has extensive white papers describing protocol components, approachable user-facing documentation, and adequate code comment coverage.	**Strong**
Front-Running Resistance	We did not identify any issues related to front-running.	**Satisfactory**
Low-Level Manipulation	Use of assembly is minimal and limited to some math and error reporting libraries.	**Satisfactory**
Testing and Verification	The system has adequate unit test coverage and includes property-based testing via the Hypothesis framework, but more end-to-end test scenarios are still needed.	**Satisfactory**

Summary of Findings

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

ID	Title	Type	Severity
1	Missing event for critical operation	Auditing and Logging	Informational
2	Missing check for new value could cause incorrect event to be emitted	Data Validation	Low
3	SafeERC20 functions not used in FeeBank	Undefined Behavior	Informational
4	Lack of two-step process for critical operations	Access Controls	Medium
5	Lack of zero address check on functions	Data Validation	Low
6	Solidity compiler optimizations can be risky	Undefined Behavior	Undetermined
7	Risk of trapped funds in the GryoTokenL1 contract due to missing ETH validation	Access Controls	Low
8	GyroTwoPool constructor allows pathological alpha/beta	Data Validation	Undetermined
9	calcOutGivenIn arithmetic allows users to extract asset for free	Data Validation	High
10	Discrepancies with error code ranges	Data Validation	Informational
11	CEMM calcInGivenOut invariant could decrease after successful call	Undefined Behavior	Undetermined

12	CPMMV-3 calcOutGivenIn invariant could decrease after successful call and give free tokens	Undefined Behavior	**Undetermined**
13	Lack of return value check when minting and redeeming could lead to unexpected results	Undefined Behavior	**High**
14	Lack of zero address check on functions	Data Validation	**Low**
15	ecrecover signature malleability	Data Validation	**Informational**
16	Duplicate pool registration could result in unexpected behavior	Data Validation	**Low**

Summary of Recommendations

The Gyroscope protocol smart contracts were reviewed on a rolling basis as they were finalized. Trail of Bits recommends that the Gyroscope protocol team address the findings shared through GitHub and take the following additional steps prior to deployment:

- Add Slither to the continuous integration pipeline of the project's GitHub repositories. To silence false-positive Slither findings, a special comment can be added in the implementation. By using this process, no pull request that contains unresolved Slither findings will be merged.
- Continue to improve test coverage by including more integration testing, negative testing scenarios, and additional system invariants to test with Hypothesis.
- Continue to work on cleaning up dead code and simplifying the system architecture where possible.
- Investigate blockchain monitoring solutions and begin developing an incident response plan.
- Implement multisig schemes for holders of privileged roles where possible, ideally backed by hardware wallets.

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
Front-Running Resistance	The system's resistance to front-running attacks
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

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 is not applicable 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. Token Integration Checklist

The following checklist provides recommendations for interactions with arbitrary tokens. Every unchecked item should be justified, and its associated risks, understood. For an up-to-date version of the checklist, see `crytic/building-secure-contracts`.

For convenience, all Slither utilities can be run directly on a token address, such as the following:

```
slither-check-erc 0xdac17f958d2ee523a2206206994597c13d831ec7 TetherToken --erc erc20
slither-check-erc 0x06012c8cf97BEaD5deAe237070F9587f8E7A266d KittyCore --erc erc721
```

To follow this checklist, use the below output from Slither for the token:

```
slither-check-erc [target] [contractName] [optional: --erc ERC_NUMBER]
slither [target] --print human-summary
slither [target] --print contract-summary
slither-prop . --contract ContractName # requires configuration, and use of Echidna
and Manticore
```

General Considerations

- ❏ **The contract has a security review.** Avoid interacting with contracts that lack a security review. Check the length of the assessment (i.e., the level of effort), the reputation of the security firm, and the number and severity of the findings.
- ❏ **You have contacted the developers.** You may need to alert their team to an incident. Look for appropriate contacts on `blockchain-security-contacts`.
- ❏ **They have a security mailing list for critical announcements.** Their team should advise users (like you!) when critical issues are found or when upgrades occur.

Contract Composition

- ❏ **The contract avoids unnecessary complexity.** The token should be a simple contract; a token with complex code requires a higher standard of review. Use Slither's `human-summary` printer to identify complex code.
- ❏ **The contract uses `SafeMath`.** Contracts that do not use `SafeMath` require a higher standard of review. Inspect the contract by hand for `SafeMath` usage.
- ❏ **The contract has only a few non-token-related functions.** Non-token-related functions increase the likelihood of an issue in the contract. Use Slither's `contract-summary` printer to broadly review the code used in the contract.

- ❏ **The token has only one address.** Tokens with multiple entry points for balance updates can break internal bookkeeping based on the address (e.g., `balances[token_address][msg.sender]` may not reflect the actual balance).

Owner Privileges

- ❏ **The token is not upgradeable.** Upgradeable contracts may change their rules over time. Use Slither's `human-summary` printer to determine whether the contract is upgradeable.
- ❏ **The owner has limited minting capabilities.** Malicious or compromised owners can abuse minting capabilities. Use Slither's `human-summary` printer to review minting capabilities, and consider manually reviewing the code.
- ❏ **The token is not pausable.** Malicious or compromised owners can trap contracts relying on pausable tokens. Identify pausable code by hand.
- ❏ **The owner cannot blacklist the contract.** Malicious or compromised owners can trap contracts relying on tokens with a blacklist. Identify blacklisting features by hand.
- ❏ **The team behind the token is known and can be held responsible for abuse.** Contracts with anonymous development teams or teams that reside in legal shelters require a higher standard of review.

ERC20 Tokens

ERC20 Conformity Checks

Slither includes a utility, `slither-check-erc`, that reviews the conformance of a token to many related ERC standards. Use `slither-check-erc` to review the following:

- ❏ **`Transfer` and `transferFrom` return a boolean.** Several tokens do not return a boolean on these functions. As a result, their calls in the contract might fail.
- ❏ **The `name`, `decimals`, and `symbol` functions are present if used.** These functions are optional in the ERC20 standard and may not be present.
- ❏ **`Decimals` returns a `uint8`.** Several tokens incorrectly return a `uint256`. In such cases, ensure that the value returned is below 255.
- ❏ **The token mitigates the known ERC20 race condition.** The ERC20 standard has a known ERC20 race condition that must be mitigated to prevent attackers from stealing tokens.

Slither includes a utility, `slither-prop`, that generates unit tests and security properties that can discover many common ERC flaws. Use `slither-prop` to review the following:

- ❏ **The contract passes all unit tests and security properties from `slither-prop`.** Run the generated unit tests and then check the properties with Echidna and Manticore.

Risks of ERC20 Extensions

The behavior of certain contracts may differ from the original ERC specification. Conduct a manual review of the following conditions:

- ❏ **The token is not an ERC777 token and has no external function call in `transfer` or `transferFrom`.** External calls in the transfer functions can lead to reentrancies.
- ❏ **`Transfer` and `transferFrom` should not take a fee.** Deflationary tokens can lead to unexpected behavior.
- ❏ **Potential interest earned from the token is taken into account.** Some tokens distribute interest to token holders. This interest may be trapped in the contract if not taken into account.

Token Scarcity

Reviews of token scarcity issues must be executed manually. Check for the following conditions:

- ❏ **The supply is owned by more than a few users.** If a few users own most of the tokens, they can influence operations based on the tokens' repartition.
- ❏ **The total supply is sufficient.** Tokens with a low total supply can be easily manipulated.
- ❏ **The tokens are located in more than a few exchanges.** If all the tokens are in one exchange, a compromise of the exchange could compromise the contract relying on the token.
- ❏ **Users understand the risks associated with a large amount of funds or flash loans.** Contracts relying on the token balance must account for attackers with a large amount of funds or attacks executed through flash loans.
- ❏ **The token does not allow flash minting.** Flash minting can lead to substantial swings in the balance and the total supply, which necessitate strict and comprehensive overflow checks in the operation of the token.

ERC721 Tokens

ERC721 Conformity Checks

The behavior of certain contracts may differ from the original ERC specification. Conduct a manual review of the following conditions:

- ❑ **Transfers of tokens to the `0x0` address revert.** Several tokens allow transfers to `0x0` and consider tokens transferred to that address to have been burned; however, the ERC721 standard requires that such transfers revert.
- ❑ **`safeTransferFrom` functions are implemented with the correct signature.** Several token contracts do not implement these functions. A transfer of NFTs to one of those contracts can result in a loss of assets.
- ❑ **The `name`, `decimals`, and `symbol` functions are present if used.** These functions are optional in the ERC721 standard and may not be present.
- ❑ **If it is used, `decimals` returns a `uint8(0)`.** Other values are invalid.
- ❑ **The `name` and `symbol` functions can return an empty string.** This behavior is allowed by the standard.
- ❑ **The `ownerOf` function reverts if the `tokenId` is invalid or is set to a token that has already been burned.** The function cannot return `0x0`. This behavior is required by the standard, but it is not always properly implemented.
- ❑ **A transfer of an NFT clears its approvals.** This is required by the standard.
- ❑ **The token ID of an NFT cannot be changed during its lifetime.** This is required by the standard.

Common Risks of the ERC721 Standard

To mitigate the risks associated with ERC721 contracts, conduct a manual review of the following conditions:

- ❑ **The `onERC721Received` callback is taken into account.** External calls in the transfer functions can lead to reentrancies, especially when the callback is not explicit (e.g., in `safeMint` calls).

- **When an NFT is minted, it is safely transferred to a smart contract.** If there is a minting function, it should behave similarly to `safeTransferFrom` and properly handle the minting of new tokens to a smart contract. This will prevent a loss of assets.
- **The burning of a token clears its approvals.** If there is a burning function, it should clear the token's previous approvals.

D. Fix Log

ID	Title	Type	Severity	Fix Status
1	Missing event for critical operation	Auditing and Logging	Informational	Fixed (PR 3)
2	Missing check for new value could cause incorrect event to be emitted	Data Validation	Low	Fixed (PR 3)
3	SafeERC20 functions not used in FeeBank	Undefined Behavior	Informational	Fixed (55c5f36)
4	Lack of two-step process for critical operations	Access Controls	Medium	Fixed
5	Lack of zero address check on functions	Data Validation	Low	Fixed (ede4599 b764671)
6	Solidity compiler optimizations can be risky	Undefined Behavior	Undetermined	Not fixed
7	Risk of trapped funds in the GryoTokenL1 contract due to missing ETH validation	Access Controls	Low	Fixed (PR 6)
8	GyroTwoPool constructor allows pathological alpha/beta	Data Validation	Undetermined	False positive
9	calcOutGivenIn arithmetic allows users to extract asset for free	Data Validation	High	Fixed
10	Discrepancies with error code ranges	Data Validation	Informational	Fixed
11	CEMM calcInGivenOut invariant could decrease after successful call	Undefined Behavior	Undetermined	Fixed

12	CPMMV-3 calcOutGivenIn invariant could decrease after successful call and give free tokens	Undefined Behavior	**Undetermined**	**Fixed**
13	Lack of return value check when minting and redeeming could lead to unexpected results	Undefined Behavior	**High**	**Fixed (d6b40e8 5cfd57c)**
14	Lack of zero address check on functions	Data Validation	**Low**	**Fixed (3fac348)**
15	ecrecover signature malleability	Data Validation	**Informational**	**Not fixed**
16	Duplicate pool registration could result in unexpected behavior	Data Validation	**Low**	**Fixed (Code removed)**