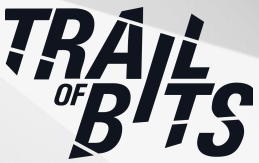

Helios Global

Security Assessment (Summary Report)

February 27, 2024

Prepared for:
William Skinner
Helios Global

Prepared by: **Richie Humphrey and Ronald Eytchison**

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, 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.
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

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and 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

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 consultants were associated with this project:

Richie Humphrey, Consultant
richie.humphrey@trailofbits.com

Ronald Eytchison, Consultant
ronald.eytchison@trailofbits.com

Project Timeline

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

Date	Event
February 1, 2024	Pre-project kickoff call
February 9, 2024	Delivery of report draft
February 12, 2024	Report readout meeting
February 27, 2024	Delivery of summary report

Executive Summary

Engagement Overview

Helios Global engaged Trail of Bits to review the security of the Helios protocol, which is a system for onboarding investments into real-world solar energy projects.

A team of two consultants conducted the review from February 5 to February 9, 2024, for a total of one engineer-week of effort. Our testing efforts focused on the pools and related contracts. With full access to the source code and documentation, we performed static and dynamic testing of Helios, using automated and manual processes.

Observations and Impact

The Helios development team developed the platform with a thoughtful attitude toward security, drawing architectural inspiration from well-known codebases. Additionally, invested funds are stored off-chain, which notably diminishes the risk of honeypot scams by limiting at-risk funds to those transitioning into or out of the protocol.

However, Helios implements a secondary accounting system for its pool tokens on top of the accounting system inherent to ERC-20. This alternate method of storing deposit balances adds complexity and increases the chance of bugs such as those we identified in TOB-HELI-1, TOB-HELI-4, TOB-HELI-7, and TOB-HELI-8, the last of which could result in locked tokens. This feature also compromises the protocol's compliance with the ERC-20 standard, potentially impacting the protocol's wider adoption and integration with other decentralized finance platforms.

There are code quality issues that require attention, including unaddressed "to-do" comments, redundant variables, and incomplete NatSpec documentation. Furthermore, project design specifications should be revisited in light of missing data validations and unexpected system behavior discovered during the review, as described in TOB-HELI-5 and TOB-HELI-7.

Recommendations

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

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.
- **Retroactively create a system specification.** The issues discovered during the review illustrate that Helios could benefit from a more formal specification of its

components. Much of the information needed to create a specification is already present in code comments or other provided documentation.

Creating a specification will allow engineers to work on new project features with higher confidence and will highlight security issues. It also allows for easy communication between the development team, the founder, and other stakeholders.

- **Create stateful invariant tests.** As part of this review, we conducted an invariant testing campaign that identified multiple issues, including TOB-HELI-1 and TOB-HELI-8. The results and the code for the test harness are reported in the invariant testing GitHub issue that we submitted to the Helios repository (TOB-HELI-Invariant-Testing). Identifying and testing system properties will expose vulnerable states and prevent engineers from inadvertently making changes that later break the properties.
- **Improve the existing fuzz tests.** While fuzzing the caller address has certain benefits, including other arguments such as amounts will greatly enhance the effectiveness of the tests. Use realistic values for amounts; the current tests use dust amounts. The code quality GitHub issue that we submitted to the Helios repository (TOB-HELI-Code-Quality) contains more detail.
- **Consider having an additional review performed.** During our time-boxed review, we found new issues every day through the last day of the engagement. This can be an indication that there are more issues in the code that were not found due to time constraints. Consider having a follow-up review performed after the system has been refactored and tests have been added. A follow-up review would help verify whether the system's security maturity has improved and may help reveal outstanding bugs that were not discovered during this assessment.

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	Although there is not much math used in the codebase, there are no tests that directly target any of the math that is used. We also identified a precision loss issue, reported in TOB-HELI-Code-Quality.	**Moderate**
Auditing	Events are properly emitted for critical operations, but no incident response plan has been created.	**Moderate**
Authentication / Access Controls	Access controls have been added to all privileged functions except for `CreateLiquidityLocker`, as noted in TOB-HELI-6. There is no clear documentation about the protocol's actors and their respective privileges in the system. Also, there are no tests to cover actor-specific privileges.	**Moderate**
Complexity Management	The secondary accounting system adds significant complexity (TOB-HELI-1, TOB-HELI-4, and TOB-HELI-7). There are unused variables, functions, and events. Finally, some function behavior is unclear, as described in TOB-HELI-5.	**Weak**
Decentralization	Critical features are controlled by a centralized entity. At any time, the admin can transfer 100% of investor funds off-chain. The amount of yield to distribute is at the discretion of the admin, and the admin can also change pool parameters.	**Weak**
Documentation	We noted missing or incomplete NatSpec comments. There is no external user-facing documentation or technical/developer documentation. However, the wiki that the Helios team began during the review week	**Weak**

	appears to be a good start to addressing these documentation issues.	
Low-Level Manipulation	Low-level manipulation is not used in this codebase.	**Not Applicable**
Testing and Verification	Current tests include unit and stateless fuzz tests. The current stateless fuzz tests could be improved (refer to TOB-HELI-Code-Quality). The test suite could be improved with stateful invariant testing and fork tests (refer to TOB-HELI-Invariant-Testing).	**Moderate**
Transaction Ordering	The current set of smart contracts is vulnerable to transaction ordering risks that could allow attackers to seize distributed yield (TOB-HELI-2) or front run pending withdrawal conclusions (TOB-HELI-3). The bug identified in TOB-HELI-8 also amplifies the effects of transaction ordering risks.	**Weak**

Summary of Findings

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

ID	Title	Type	Severity
1	TOB-HELI-1: Reinvesting yield permanently locks tokens	Data Validation	High
2	TOB-HELI-2: Yield distribution is susceptible to front-running	Timing	High
4	TOB-HELI-4: Unbounded array perpetually increases gas cost	Data Validation	High
8	TOB-HELI-8: Accounting bug in withdraw enables sandwich and flashloan attacks	Data Validation	High
5	TOB-HELI-5: Unfilled regional pools deviate from expected behavior	Data Validation	Medium
6	TOB-HELI-6: Missing access control on CreateLiquidityLocker	Access Controls	Low
3	TOB-HELI-3: Pending withdrawal conclusions may be front run	Timing	Low
7	TOB-HELI-7: Withdrawals revert when multiple deposit instances are submitted	Data Validation	Low

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 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. 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 February 23, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the Helios 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 eight issues described in this report, Helios has resolved six issues, has partially resolved two issues, and has not resolved the remaining issue. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Reinvesting yield permanently locks tokens	**Resolved**
2	Yield distribution is susceptible to front-running	**Resolved**
3	Pending withdrawal conclusions may be front run	**Partially Resolved**
4	Unbounded array perpetually increases gas cost	**Resolved**
5	Unfilled regional pools deviate from expected behavior	**Partially Resolved**
6	Missing access control on CreateLiquidityLocker	**Resolved**
7	Withdrawals revert when multiple deposit instances are submitted	**Resolved**
8	Accounting bug in withdraw enables sandwich and flashloan attacks	**Unresolved**

Detailed Fix Review Results

TOB-HELI-1: Reinvesting yield permanently locks tokens
Resolved in commit `b5b28ea`. This workflow and the related function have been removed, so this vulnerability no longer exists.

TOB-HELI-2: Yield distribution is susceptible to front-running
Resolved in commit `50a7a0a`. The introduction of the new `finalized` state mitigates this risk as long as the pool status is set to `finalized` prior to the yield distribution.

TOB-HELI-3: Pending withdrawal conclusions may be front run
Partially resolved in commit `e894498`. Adding the mechanism to request funds from the blended pool potentially reduces the amount of pending withdrawals that will be created; however, this issue can still be triggered. We also note that this issue can be circumvented if the admin sends tokens to the contract in the same transaction that the withdrawal is concluded.

We recommend adding documentation around this issue to assist future development.

TOB-HELI-4: Unbounded array perpetually increases gas cost
Resolved in commit `50a7a0a`. The use of OpenZeppelin's `EnumerableSet.AddressSet` allows for real-time O(1) lookups.

TOB-HELI-5: Unfilled regional pools deviate from expected behavior
Partially resolved in commit `50a7a0a`. The `calculateYield` function was fixed, and a new `finalized` state was introduced, which addresses the late deposit issue. However, this state is not checked when users repay or borrow funds, which means that when the admin makes a mistake, funds can be moved in and out of the pool prematurely.

TOB-HELI-6: Missing access control on CreateLiquidityLocker
Resolved in commit `2abba53`. The concept of the liquidity locker was removed, so this issue no longer exists.

TOB-HELI-7: Withdrawals revert when multiple deposit instances are submitted
Resolved in commit `e894498`. Withdrawals no longer accept indices as parameters, nor do they mutate deposit instances.

TOB-HELI-8: Accounting bug in withdraw enables sandwich and flashloan attacks
Unresolved. The Helios team attempted to resolve this issue as part of commit `e894498`; however, this change introduced a new, similar vulnerability.

Now, the lockup period can be skipped by transferring the tokens to another user.

Here is an example exploit scenario:

1. Eve creates a deposit for the minimum amount using address 1.

2. Time passes, and the lockup period for address 1 expires.
3. Now, Eve can create additional deposits using an alternate address 2 for up to the maximum investment amount, including through the use of a flash loan if she wants.
4. She can evade the lockup period by calling `ERC20.transfer` on the pool tokens to transfer them from address 2 to address 1.
5. She can then immediately withdraw all tokens from address 1. The current logic will check and see that address 1 is a valid holder and does not have any tokens in lockup, so it will allow all of the tokens to be withdrawn, circumventing the lockup period.

One way to mitigate this would be to override the ERC-20 `transfer` and `transferFrom` functions and have the transaction revert if the amount is more than the value of `unlockedToWithdraw`.

```
abstract contract AbstractPool is ERC20, ReentrancyGuard {

    function transfer(address to, uint256 amount) external override returns (bool) {
        require(amount <= unlockedToWithdraw(msg.sender), "TOKENS LOCKED");
        return super.transfer(to, amount);
    }

    function transferFrom(address from, address to, uint256 amount) external
override returns (bool) {
        require(amount <= unlockedToWithdraw(from), "TOKENS LOCKED");
        return super.transferFrom(from, to, amount);
    }
```

Figure C.1: An example of how to override the `transfer` and `transferFrom` functions and add logic to respect the lockup period

D. 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.