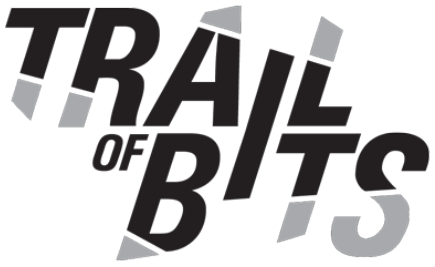

Thesis

Security Assessment

June 8, 2020

Prepared For:
Antonio Salazar Cardozo | *Thesis*
antonio.salazarcardozo@thesis.co

Matt Luongo | *Thesis*
matt@thesis.co

Piotr Dyraga | *Thesis*
piotr.dyraga@thesis.co

Prepared By:
Jim Miller | *Trail of Bits*
james.miller@trailofbits.com

Johanna Ratliff | *Trail of Bits*
johanna.ratliff@trailofbits.com

Sam Sun | *Trail of Bits*
sam.sun@trailofbits.com

Assessment Summary

From May 18 to June 5, 2020, Trail of Bits performed an assessment of the `keep-common`, `keep-core`, `keep-ecdsa`, `sortition-pools`, and `tbtc` repositories. This report describes the work performed on the `tbtc` repository. Trail of Bits reported numerous findings, with various levels of severity. The description of classifications for all of our findings can be found in Appendix A.

The assessment was scoped to apply manual review to `tBTC` and its architecture, targeting smart contracts written in Solidity. We reported 13 issues in tBTC, and each one has been logged as an issue in their public repository.

Of the high-severity findings discovered, two findings would allow an attacker to seize signer bonds, and another finding would allow governance to bypass the delay when changing settings. In addition to these issues, we report some informational findings, which present no immediate threat but prompt recommendations to strengthen the system.

Additionally, a severe vulnerability was discovered immediately prior to our engagement, and we assisted in the characterization and remediation of it.

Thesis must improve their testing and verification. Some of our findings could have been prevented with more rigorous unit testing. Other findings focused on edge cases and potentially malicious behavior, such as this finding concerning fee rebates, and we therefore recommend that Thesis document malicious behaviors the system should protect against, including edge cases that have not been accounted for. Once these behaviors have been identified, property testing can be used to ensure the system is protected against these attacks.

Overall, we achieved strong coverage of all the codebases in scope. We recommend Thesis address all the issues discovered and review their fixes and any other subsequent changes in another assessment.

Lastly, we reviewed the maturity of the codebase and the likelihood of future issues. For each control family we rate the maturity from strong to missing, and give a brief explanation of our reasoning.

Code Maturity Evaluation

Category Name	Description
Access Controls	**Moderate.** Some findings were mitigated by introducing additional access controls. However, insufficient access controls allowed us to bypass the governance delay.
Arithmetic	**Further Investigation Required.** No issues were discovered relating to arithmetic. Arithmetic was not a primary area of focus of this report.
Assembly Use	**Strong.** There was minimal use of assembly in the codebase, and we report no related issues.
Centralization	**Moderate.** We reported one finding in which a rogue owner could select their own Keep factory, but we did not find many issues related to centralization.
Contract Upgradeability	**Not Applicable.** The proxy pattern is used instead of traditional upgradeability.
Function Composition	**Moderate.** The Deposit code was well organized. However, other design decisions relating to the redemption fee calculations made the code difficult to understand.
Front-Running	**Not Considered.** A previous assessment by ConsenSys addressed front-running concerns around ECDSA fraud submissions.
Monitoring	**Further Investigation Required.** No issues were discovered relating to monitoring. However, if this is an area of concern, it should be investigated further.
Specification	**Moderate.** The high-level documentation was comprehensive but slightly out of date. As mentioned above, some findings could have been mitigated by detailing potential malicious behavior more thoroughly.
Testing & Verification	**Weak.** Thesis provided many tests, but since our findings could have been prevented with more thorough testing, it should be improved.

Project Dashboard

Commit hashes of the reviewed repositories:

- `keep-common: 9fbd0b9c5ad2376ee49b3380e038648d87f0b103`
- `keep-core: 16554512ae545608a5e902160949defb626cc3bd`
- `keep-ecdsa: f52ec8f65d3aa99529fd48b182069a78b5d473f3`
- `sortition-pools: c0b2c7d04125176cad614d3b5a458858dfbd25a6`
- `tbtc: b823795fd5c870364947474d8495b12102812b74`

Appendix A. Vulnerability Classifications

Vulnerability Classes	
Class	Description
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices, or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing system failure
Documentation	Related to documentation accuracy
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Timing	Related to race conditions, locking, or order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	Description
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Undetermined	The extent of the risk was not determined during this engagement
Low	The risk is relatively small or is not a risk the customer has indicated is important
Medium	Individual user's information is at risk, exploitation would be bad for client's reputation, moderate financial impact, possible legal

	implications for client
High	Large numbers of users, very bad for client's reputation, or serious legal or financial implications

Difficulty Levels	
Difficulty	Description
Undetermined	The difficulty of exploit was not determined during this engagement
Low	Commonly exploited, public tools exist or can be scripted that exploit this flaw
Medium	Attackers must write an exploit, or need an in-depth knowledge of a complex system
High	The attacker must have privileged insider access to the system, may need to know extremely complex technical details, or must discover other weaknesses in order to exploit this issue

Appendix B. Code Maturity Classifications

Code Maturity Classes	
Category Name	Description
Access Controls	Related to the authentication and authorization of components.
Arithmetic	Related to the proper use of mathematical operations and semantics.
Assembly Use	Related to the use of inline assembly.
Centralization	Related to the existence of a single point of failure.
Upgradeability	Related to contract upgradeability.
Function Composition	Related to separation of the logic into functions with clear purpose.
Front-Running	Related to resilience against front-running.
Key Management	Related to the existence of proper procedures for key generation, distribution, and access.
Monitoring	Related to use of events and monitoring procedures.
Specification	Related to the expected codebase documentation.
Testing & Verification	Related to the use of testing techniques (unit tests, fuzzing, symbolic execution, etc.).

Rating Criteria	
Rating	Description
Strong	The component was reviewed and no concerns were found.
Satisfactory	The component had only minor issues.
Moderate	The component had some issues.
Weak	The component led to multiple issues; more issues might be present.
Missing	The component was missing.

Not Applicable	The component is not applicable.
Not Considered	The component was not reviewed.
Further Investigation Required	The component requires further investigation.