SOLIDIFIED

Audit Report for GEB Foundation - September 12, 2022

Summary

Audit Report prepared by Solidified covering the RAI Ungovernance smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on September 12, 2022, and the results are presented here.

Audited Files

The source code has been supplied in the following public source code repositories:

https://github.com/reflexer-labs/geb-tellor-median/blob/master/src/TellorRelayer.sol
https://github.com/reflexer-labs/geb-gov-minimization-overlay/blob/master/src/overlays/minimal/MinimalOSMOverlay.sol

Commit numbers:

```
TellorRelayer.sol: 75dbaf5d1e82e323538b2146f6a03b3ab6ebb0d7
MinimalOSMOverlay.sol: 34baff218486952b85a76b8185b6b73ee5d182ef
```

Update: Fixes received on September 14, 2022.
New commit numbers:

```
TellorRelayer.sol: fb67dab5fae136ac6137550f68bd1266f1b4e309
MinimalOSMOverlay.sol: 34baff218486952b85a76b8185b6b73ee5d182ef
```

Intended Behavior

RAI Ungovernance is a set of smart contracts that limit the amount of power governance has over a GEB instance.

Findings

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	Low	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	High	-

Issues Found

Solidified found that the RAI Ungovernance contracts contain no critical issues, no major issues, 2 minor issues, 3 informational notes and 1 warning.

We recommend issues are amended, while informational notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	TellorRelayer.sol: Validation mismatch between function modifyParameters() and the contract's constructor	Minor	Resolved
2	TellorRelayer.sol: Function removeAuthorization() can be front run by a malicious authorized account	Minor	Acknowledged
3	TellorRelayer.sol: The authorizedAccounts mapping values are unnecessarily declared as uint	Note	Acknowledged
4	TellorRelayer.sol: Function getResultWithValidity() reverts if getCurrentValue fails	Note	Acknowledged
5	TellorRelayer.sol: Redundant check for address(0) of tellor	Note	Resolved
6	MinimalOSMOverlay.sol: An authorized account could utilize the swapOracle() function to execute a transaction sandwiching attack	Warning	Acknowledged

Critical Issues

No critical issues have been found.

Major Issues

No major issues have been found.

Minor Issues

1. `TellorRelayer.sol`: Validation mismatch between function `modifyParameters()` and the contract’s constructor

The function `modifyParameters()` validates `staleThreshold` to be greater than 1, while the contract’s constructor only ensures it is greater than zero.

Recommendation

Unify validation between `modifyParameters()` and the constructor, preferably using a single setter function.

Status

Resolved

2. TellorRelayer.sol: Function removeAuthorization() can be front run by a malicious authorized account

The function `removeAuthorization()` can always be front run by the account it is trying to remove.

Recommendation

Consider limiting access to `removeAuthorization()` to only a contract owner or a super authorized admin.

Status

Acknowledged. Team's response: "*We are aware of this possibility, but in practice this cannot happen, as all `authorizedAccounts` are smart contracts (we allow no privileged EOAs on the contract before attaching it to the system). The only address authed in this one will be `Pause.proxy()`, our timelock contract, controlled by governance*".

Informational Notes

3. TellorRelayer.sol: The authorizedAccounts mapping values are unnecessarily declared as uint

The `authorizedAccounts` mapping only needs a value of either `true` or `false`.

Recommendation

Consider redeclaring `authorizedAccounts` as `(address => bool)` in order to save on gas fees.

Status

Acknowledged. Team's response: "*All our contracts follow this pattern so we chose to continue using uint for authorized accounts to maintain consistency. Interestingly, storing it as an uint is actually cheaper (gaswise) than using a boolean*".

4. TellorRelayer.sol: Function getResultWithValidity() reverts if getCurrentValue fails

Function `getResultWithValidity()` reverts if `getCurrentValue` fails. Consider if it would be more consistent to return `false` in such a case (for instance, the function returns `false` if the `tellor` address is not set).

Status

Acknowledged. Team's response: "*We are using Tellor's standard contract to fetch the data, including a try/catch statement would entail modifying the UsingTelor contract, something that we do not want to do, as that contract has been audited and battle tested by numerous protocols by now, and additionally the failure state (the call reverting instead of returning `valid == false`) is non critical*".

5. TellorRelayer.sol: Redundant check for address(0) of tellor

In the read() and getResultwithValidity() functions, there are checks for address(0) for tellor. This check is redundant as it is already done in the constructor.

Recommendation

Consider removing these checks to avoid redundancy and save on gas fees.

Status

Resolved

Warnings

6. MinimalOSMOverlay.sol: An authorized account could utilize the swapOracle() function to execute a transaction sandwiching attack

If one of the `trustedOracles` was malfunctioning, a compromised authorized account could make an economic attack on the system by sandwiching a `swapOracle()` function call in-between some other financial actions.

The current safeguards do not allow removing a malfunctioning oracle quickly and any authorized account can enable it back.

Status

Acknowledged. Team's response: "*The only authorized account on the contract will be the governance timelock, so for an oracle swap to happen a vote (~3 days) and a 24 hour timelock (in Pause) are necessary. As stated in issue #2 we will not allow any privileged EOAs before attaching this contract to the system*".

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of the GEB Foundation or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Oak Security GmbH