AAVE

GHO Stablecoin AIP

Smart Contract Security Assessment Report

Version: 1.3

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the GHO smart contracts. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the GHO smart contracts contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the AAVE smart contracts.

Overview

GHO is a collateral backed stablecoin designed to maintain a stable value in spite of market volatility. GHO can be natively integrated into the AAVE Protocol.

The purpose of this review is to validate the governance payload for the deployment of the `GHO` stablecoin.

Security Assessment Summary

This review was conducted on the files hosted on the aave/gho-aip and were assessed at commit ee6b6d0.

Note: the OpenZeppelin libraries and dependencies were excluded from the scope of this assessment.

The manual code review section of the report is focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the contracts. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout). Additionally, the manual review process focused on all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [1, 2].

To support this review, the testing team used the following automated testing tools:

- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`

Output for these automated tools is available upon request.

Findings Summary

The testing team identified a total of 3 issues during this assessment. Categorised by their severity:

- High: 1 issue.
- Medium: 1 issue.
- Informational: 1 issue.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the AAVE smart contracts. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as “informational”.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
GHO-AIP-01	`GhoListingPayload` permanently modifies the storage of the governance executor contract	High	Resolved
GHO-AIP-02	Contracts can be created before execution, causing the payload to fail	Medium	Resolved
GHO-AIP-03	Miscellaneous General Comments	Informational	Resolved

GHO-AIP-01	`GhoListingPayload` permanently modifies the storage of the governance executor contract		
Asset	`GhoListingPayload.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description

Executing the `GhoListingPayload` payload will change the storage of the governance executor contract, which could affect future payloads.

Most of the variables in `GhoListingPayload` are either `constant` or `immutable`, which do not affect the contract's storage.

However, `GhoListingPayload` does contain two state variables: `GHO_TOKEN` and `GHO_FLASHMINTER`. The `execute()` function modifies both of these variables. If `execute()` is called in a standard `CALL`, these values would be stored in the storage of `GhoListingPayload` itself. As it is intended to be called by `DELEGATECALL` when executed, it means that the function would modify the storage of the `Executor` contract.

This will not prevent `execute()` from functioning properly, however it will store the values of `GHO_TOKEN` and `GHO_FLASHMINTER` in the first two storage slots of the `Executor` contract. This creates a permanent hazard. When the next payload is executed and accesses the storage slots of the `Executor` contract, then it would have unexpected values in those slots which would produce unexpected results.

Recommendations

Do not declare `GHO_TOKEN` and `GHO_FLASHMINTER` as state variables. Instead, consider declaring them as `immutable` variables within the constructor. The values are deterministic and can be calculated at construction time.

Resolution

The development team implemented the recommendation at 957b050.

GHO-AIP-02	Contracts can be created before execution, causing the payload to fail		
Asset	`GhoListingPayload.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

A Denial of Service attack exists which may prevent payload execution. Either of the contracts `GhoToken` or `GhoFlashMinter` may be deployed before the execution payload attempts to create these contracts. As a result, the payload execution will revert.

`execute()` calls the contract at `0x2401ae9bBeF6745836 2710f90302Eb52b5Ce835a` to deploy new contracts using `CREATE2`. This technique means that the address of the created contract is deterministic and does not depend on the address of the caller. Since the `0x2401ae9bBeF6745836 2710f90302Eb52b5Ce835a` contract is permissionless any user may front-run the executor call with the same parameters. This will deploy the contract that the payload would deploy, at the same address.

This would not cause any major security vulnerability in the deployed contracts themselves, as their essential configurations are in their constructors, and so included in their deployment bytecode. However, it would cause the payload itself to fail. `_deployCreate2()` would revert when it calls `0x2401ae9bBeF6745836 2710f90302Eb52b5Ce835a` since the contract address already contains bytecode.

To resolve the issue a new payload would need to be deployed, and a second vote must pass to execute the changes.

Recommendations

Consider checking for the contracts in advance at the predicted addresses in `GhoListingPayload` and not deploying if they are already there. In this case, consider also checking that the contract's code is as expected.

Resolution

The predicted addresses are checked to ensure that they have no code at 957b050.

GHO-AIP-03	Miscellaneous General Comments
Asset	`contracts/*`
Status	**Resolved:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings that do not have direct security implications:

1. **Zero address checks:** Consider adding checks for `address(0)` in the following places:
   - `GhoListingPayload` `constructor` all parameters.
2. **Payload numbering:** In the `GhoListingPayload` comment section headings, there are two number 6 steps.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The development team have acknowledged these findings, addressing them where appropriate.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `Foundry` framework was used to perform these tests and the output is given below.

```
Running 1 test for tests/GhoPayloadSigPTest.t.sol:GhoPayloadSigPTest
[PASS] testSigpChecks() (gas: 21819315)
Test result: ok. 1 passed; 0 failed; finished in 558.22ms
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

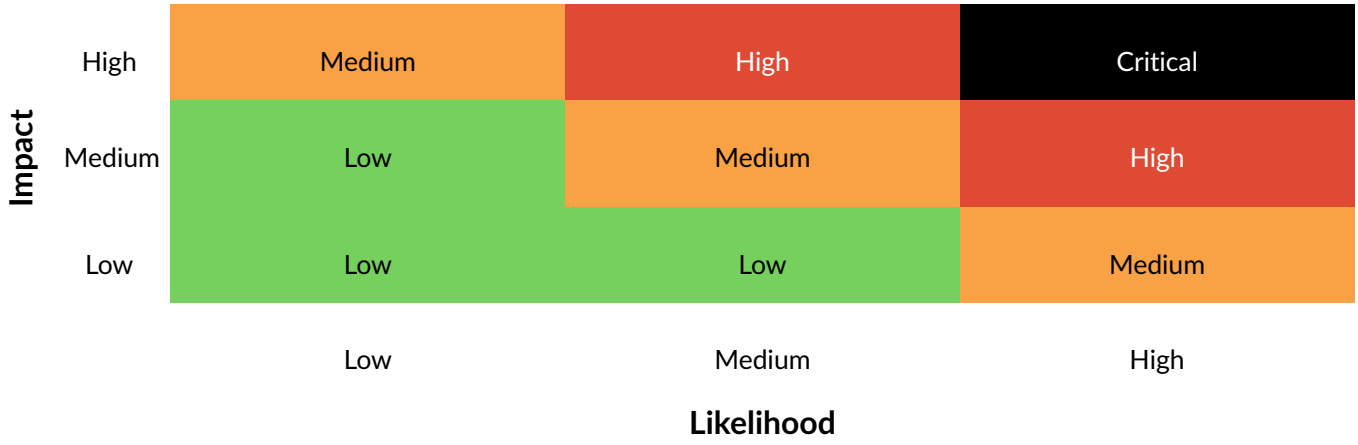

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: https://blog.sigmaprime.io/solidity-security.html. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: http://www.dasp.co/. [Accessed 2018].