Aave

Gho Steward Contract Review

Version: 1.0

June, 2023

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the `GhoSteward` smart contract. 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 `GhoSteward` smart contract. 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 `GhoSteward` smart contract.

Overview

The GHO Steward is a smart contract that is capable of adjusting the predefined `GhoToken` parameters for Aave V3. This contract can increase Aave Protocol bucket capacity or change the variable borrow rate.

Security Assessment Summary

This review was conducted on the smart contract `GhoSteward.sol` hosted on the gho-core repository and were assessed at commit 2dcaadd.

A subsequent round of testing targeted commit 735cf12 and focused solely on verifying whether the previously identified issues had been resolved.

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`
- Surya: `https://github.com/ConsenSys/surya`

Output for these automated tools is available upon request.

Findings Summary

The testing team identified a total of 2 issues during this assessment. Categorised by their severity:

- Informational: 2 issues.

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
AGS-01	Only Zero Values Are Accepted If The Previous Parameters Are Zero	Informational	**Closed**
AGS-02	Miscellaneous General Comments	Informational	**Resolved**

AGS-01	Only Zero Values Are Accepted If The Previous Parameters Are Zero
Asset	`GhoSteward.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

If the previous values of borrow rate or bucket capacity is ever zero, they can never be set to a non-zero value.

The functions `updateBucketCapacity()` and `updateBorrowRate()` are used respectively to increase the `bucketCapacity` and change the variable borrow rate for the Gho token in the Aave V3 protocol.

When updating values, restrictions are placed on the difference between the old and new value. This is done using a require statement that calls the functions `_bucketCapacityIncreaseAllowed()` and `_borrowRateChangeAllowed()`. These functions check for the old values of these parameters and compare them with the new ones. However, if the old values of these parameters are zero, the new values can never be set to a non-zero value.

The cause is in the functions `_bucketCapacityIncreaseAllowed()` and `_borrowRateChangeAllowed()`, which respectively check if the difference between the new capacity and the old capacity is within the max of 100% and if the difference between the old and new rate is within the max of 0.5%:

```
165 function _bucketCapacityIncreaseAllowed(uint256 from, uint256 to) internal pure returns (bool) {
      return to >= from && to - from <= from;
167  }
```

```
152 function _borrowRateChangeAllowed(uint256 from, uint256 to) internal pure returns (bool) {
      return
154     from < to
          ? to - from <= from.percentMul(BORROW_RATE_CHANGE_MAX)
156       : from - to <= from.percentMul(BORROW_RATE_CHANGE_MAX);
    }
```

Hence, if `from` parameter is zero, these functions would only return true if the `to` is also a zero value.

Recommendations

Ensure this behavior is intended.

Resolution

The development team has clarified that this behavior is intended and if a parameter is set to zero, the DAO would intervene to set non-zero values via AIP.

AGS-02	Miscellaneous General Comments
Asset	`contracts/*`
Status	**Resolved:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. `AAVE_SHORT_EXECUTOR` is hardcoded and `immutable`, consider instead using `constant`.
2. For consistency, use `<` instead of `<=` in the require statement on line [74] `require(block.timestamp <= _stewardExpiration, 'STEWARD_EXPIRED')`,
3. As an optimisation since the `bucketCapacity` is intended only to increase, use `to > from` instead of `to >= from` in `_bucketCapacityIncreaseAllowed()`.

Recommendations

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

Resolution

The comments above have been acknowledged by the development team, and relevant changes actioned in 735cf12 where appropriate.

Appendix A Test Suite

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

```
test_constructor                                        PASSED  [7%]
test_update_borrow_rate                                 PASSED  [15%]
test_update_borrow_rate_expired_time                    PASSED  [23%]
test_update_borrow_rate_debounce_time_not_respected     PASSED  [30%]
test_update_borrow_rate_invalid_rate                    PASSED  [38%]
test_update_borrow_rate_invalid_caller                  PASSED  [46%]
test_update_bucket_capacity                             PASSED  [53%]
test_update_bucket_capacity_expired_time                PASSED  [61%]
test_update_bucket_capacity_debounce_time_not_respected PASSED  [69%]
test_update_bucket_capacity_invalid_capacity            PASSED  [76%]
test_update_bucket_capacity_invalid_caller              PASSED  [84%]
test_extend_steward_expiration                          PASSED  [92%]
test_extend_steward_expiration_invalid_caller           PASSED  [100%]
```

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.

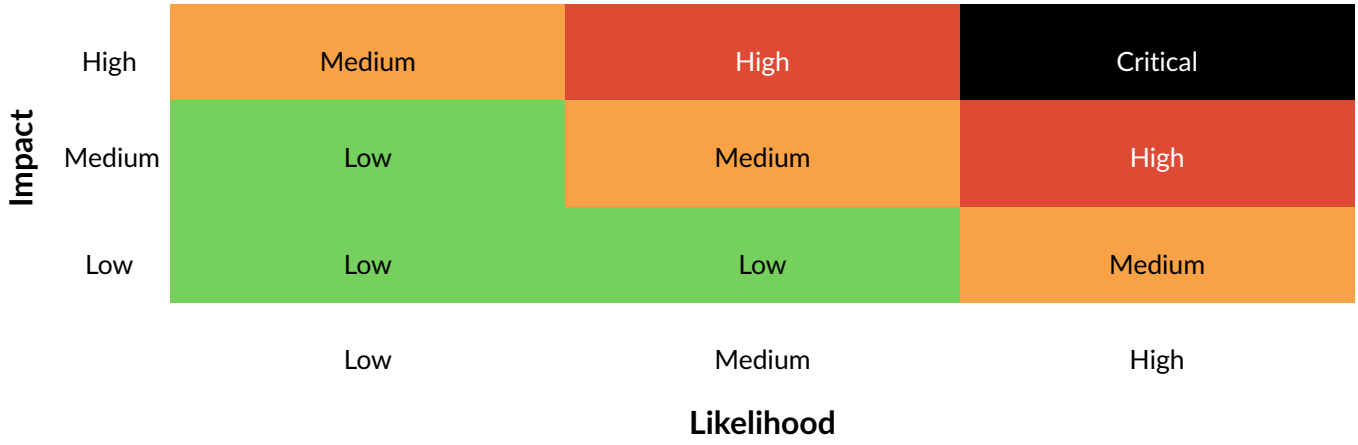

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

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].