Havven Australia Ltd

Havven Contract Review

Version: Audit 4 — IssuanceController

June, 2018

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the `IssuanceController.sol` file, which specifies a new component of the Havven system.

This review focuses solely on the security aspects of the Solidity implementation of the new component. The more general economic structure of the platform and related economic game theoretic attacks on the platform are outside the scope of this review.

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 contracts. 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 introduces the project and provides an overview of the functionality of the contracts contained within the scope of the security review. A summary of the discovered vulnerabilities is then provided, followed by a detailed review in which each vulnerability is assigned (i) a severity rating (see Vulnerability Severity Classification), (ii) an open/closed status and (iii) a recommendation. Additional findings which do not have direct security implications but are of potential interest are marked as "informational". The outputs from automated tests that were developed during the assessment are also included for reference (see Test Suite).

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

Project Overview

The contract `IssuanceController` is the subject of this review. Analysis of the contract is only considered within the context of the Havven system of smart contracts (Nomin, Havven, etc.) and not as a component of any other system.

This review is solely focused on the security of the `IssuanceController` contract, not the other contracts in the Havven system (e.g., Nomin, Havven, etc.).

Contract Overview

Note: This document is based upon a time-boxed analysis of the underlying smart contracts. Statements are not guaranteed to be accurate and do not exclude the possibility of undiscovered vulnerabilities.

The `IssuanceController` contract provides the following primary functionalities:

- Allows users to purchase `nomin` tokens by sending `eth` to the contract.
- Allows users to purchase `havven` tokens by sending `nomins` to the contract.

- Implements multiple `owner` privileges, such as:
  - Specifying the oracle that informs the contract of the USD value of `eth` and `havven` tokens.
  - Changing the address of the Havven and/or Nomin contracts with which `IssuanceController` interacts.
  - Updating the `priceStalePeriod`, which determines the duration for which the contract relies on the previously acquired USD value of `eth` and `havvens`.

Review Summary

This review was conducted on commit 584eeec404af5166dca125f904ee4a8a7c9c3b8c. Within this commit only the following contracts were reviewed:

```
└── contracts
    └── IssuanceController.sol
```

Per-Contract Vulnerability Summary

Issuance Controller (`IssuanceController.sol`)

- The purchase of `havven` and `nomin` tokens via `IssuanceController` is vulnerable to front-running attacks that disadvantage the token purchaser.
- The absence of checks on the addresses of Havven and Nomin contracts associated with `IssuanceController` allows the `owner` to receive `eth` whilst issuing illegitimate tokens.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the *Havven* 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”.

All vulnerabilities in this review have been resolved with comments from the author.

Summary of Findings

ID	Description	Severity	Status
HAV-01	The purchase of `nomins` and/or `havvens` is vulnerable to front-running	Medium	Resolved
HAV-02	Tokens purchased via `IssuanceController` may be illegitimate	Medium	Resolved
HAV-03	Miscellaneous general comments	Informational	Resolved

HAV-01	The purchase of `nomins` and/or `havvens` is vulnerable to front-running		
Asset	IssuanceController.sol		
Status	Closed: See author's response		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

Front-running [1] attacks occur when users watch a blockchain for transactions, then submit transactions with higher gas prices to give their transactions a greater order preference. By modifying the state in some way, the user is able to affect the functioning of earlier-sent transactions.

The `IssuanceController` contract is vulnerable to front-running attacks as follows. The contract uses an oracle to set the `usdToEthPrice` via the function `updatePrices`. The owner of the oracle can watch the blockchain for submitted transactions in which users send `eth` to purchase `nomins`. Upon observing submission of such a transaction, the oracle can call `updatePrice` to change the `usdToEthPrice` to the disadvantage the user. If the transaction submitted by the oracle is endowed with a large quantity of gas, it will be given priority over the original transaction. Consequently the `usdToEthPrice` can be changed prior to processing the users transaction, such that the user receives less `nomins`.

The purchase of `havvens` by the function `exchangeNominsForHavvens` is similarly vulnerable to front-running attacks involving an unfavourable change in the value of `usdToHavPrice`.

Recommendations

There are a number of ways to resolve this issue. For example, the functions called by users to purchase `nomins` or `havvens` could be modified to allow users to input the `numberOfTokensRequested`. A check could then be implemented, prior to executing the purchase, to ensure the user will receive the requested number of tokens. Although not user-friendly, users could be advised to implement their own "price locking" contract which will only execute a transaction given a specified price. Other solutions are possible.

Author's Response

The owner of the oracle is the Havven foundation. Exploit of this vulnerability would require us to watch for orders, then front run the price variable. We have a vested interest in ensuring the stability and long term viability of the network, and attacks such as this would undermine the credibility of the network. Since the primary UI users will use to exchange is our own dApp, we were unable to prioritise adding this functionality to the smart contract itself. Users are placing a large amount of trust in the Havven foundation for other functionality in the contracts, so this felt in line with that current expectation from the community to us. It is also envisaged that

at some point in the future we may be able to decentralise the oracle, which would entirely mitigate this attack vector.

HAV-02	Tokens purchased via `IssuanceController` may be illegitimate		
Asset	IssuanceController		
Status	Closed: See author's response		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

Within the set of contracts that implement the full functionality of the Havven system, balances for `havven` and `nomin` tokens are maintained in associated `TokenState` contracts. Havven and Nomin contracts interact with these `TokenState` contracts to, e.g., implement token transfers and modify user balances, while users interact with proxy contracts. This allows the `owners` to interchange the Havven and Nomin contracts in order to update the system's functionality in the future.

The `owner` of `IssuanceController` can change the addresses of the Havven and Nomin contracts associated with the `IssuanceController` contract at any stage via the `setHavven` and `setNomin` functions. There are currently no checks to ensure that the new Havven and Nomin contracts refer to the associated `TokenState` contracts. A malicious `owner` could therefore substitute the Havven and Nomin contracts, such that `eth` could be collected via `IssuanceController` without actually issuing valid `havven` or `nomin` tokens (i.e., tokens that are registered by the underlying `TokenState` contracts).

Recommendations

Checks could be implemented to provide users with some assurance that tokens purchased via `IssuanceController` are legitimate. `TokenState` contracts have an `associatedAddress`, which refers to the address of the Nomin or Havven contract endowed with the power to modify balances in the `TokenState`. `IssuanceController` could check that the addresses for the Nomin and Havven contracts associated with `IssuanceController` refer to the `associatedAddress` for the respective `TokenState` contracts.

Implementing this check would provide minor constraints on future topologies for the set of Havven contracts. The check is also imperfect as a malicious `owner` could refer both the `TokenState` and `IssuanceController` contracts to a fraudulent address or arbitratily modify the `Nomin` and `Havven` contracts. However, such activity would render the entire Havven system inactive, amplifying the consequences for malicious `owner` activity and thereby disincentivising bad behaviour.

Author's Response

Implementing this check would have made future updates to the contracts more difficult for the Havven foundation to carry out. Since our contracts are already implemented behind proxies, allowing us to change them

at will, we did not believe the additional complexity yielded benefits to the community beyond the checks that are already in place. We have a vested interest in ensuring users only receive valid tokens when interacting with the issuance controller, and users can verify that tokens they receive are valid. We believe on a cost vs benefit analysis that this check wasn't worth the additional complexity.

HAV-03	Miscellaneous general comments
Asset	IssuanceController
Status	Closed: See author's responses
Rating	Informational

Description

This section details miscellaneous informational aspects found within the contract. These items are understood to not pose immediate security risks.

- **No checks on** `priceStalePeriod` **-** The `owner` can set the state variable `priceStalePeriod` to arbitrary values by calling the function `setPriceStalePeriod` on line [160]. There are no checks to ensure that the stale period remains within a desired range - inadvertently (or intentionally) setting `priceStalePeriod` to small values could block some of the contracts functionality via the `pricesNotStale` modifier.

  ***Author's response**: The 'priceStalePeriod' variable has been left flexible intentionally.*

- **Varying Nomin/USD price -** There is an implicit assumption that one nomin is always worth one US dollar. For example, the Havven price is specified in US dollars, however Nomins are used to obtain the Havvens. We suggest using the "eUSD" term instead of "USD".

  ***Author's response**: The 'eUSD' contract will eventually be retired, while nomins are 'nUSD'. The wording was confusing when trying to name variables, and the oracle is actually retrieving the pricing in USD, not 'nUSD' as 'nUSD' is not an active trading pair on exchanges yet. This contract will always treat 'nUSD' as equivalent to 'USD', hence the naming.*

- **Ambiguous variable name -** the `exchangeNominsForHavvens()` function takes a parameter named `amount`. It is initially unclear as to whether `amount` refers to Nomins or Havvens.

  ***Author's response**: The 'amount' variable could definitely have been named more specifically.*

Recommendations

Ensure these are as expected.

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 `truffle` framework was used to perform these tests and the output is given below.

```
Contract: IssuanceController
  constructor
    ✓ sets public variables (1065ms)
  self-destruct
    ✓ can destruct from owner (1661ms)
    ✓ wont destruct from non-owner (763ms)
  stale period
    ✓ has a default stale period of 3 hours (686ms)
    ✓ will go stale with no price update after 3 hours (1200ms)
    ✓ will go stale given a stale time of 24 hours (1378ms)
  token management
    ✓ can withdraw nomins and havvens (1067ms)
  eth to nomins exchange
    ✓ will exchange at expected rate for various scenarios via fallback
 (7096ms)
    ✓ will exchange at expected rate for various scenarios via direct
 method (6902ms)
  nomins to havvens exchange
    ✓ will exchange at expected rate for various scenarios (5107ms)
    ✓ will exchange at expected rate for various scenarios given shifting
 prices (6499ms)
    ✓ will revert given a zero usd/havven price (972ms)


12 passing (34s)
```

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.

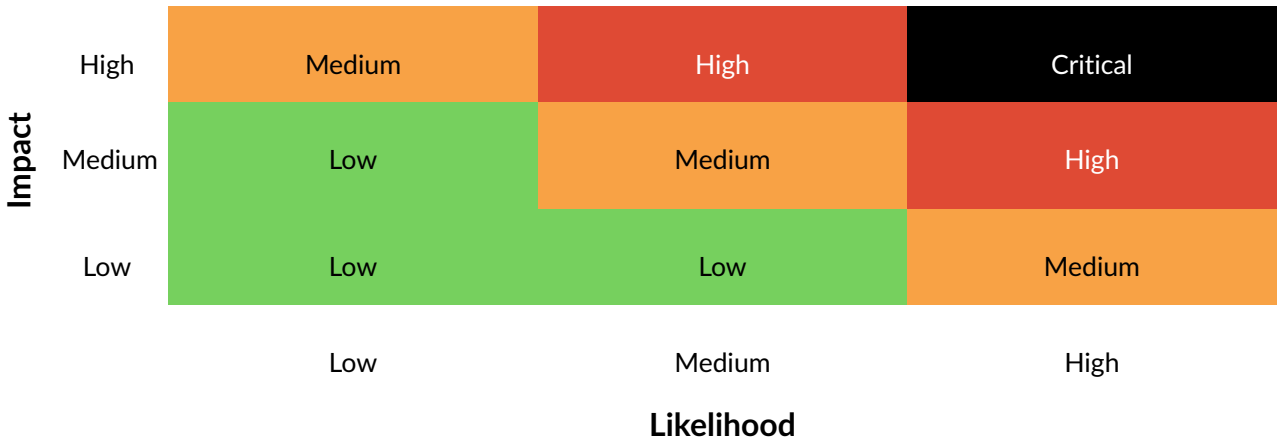

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Ethereum Smart Contract Best Practices - Front Running. Website, Available: https://consensys.github.io/smart-contract-best-practices/known_attacks/#transaction-ordering-dependence-tod-front-running.