# Virtuals Protocol Contracts
## Security Review


Cantina Managed review by:

**Joran Honig**, Lead Security Researcher
**Kankodu**, Security Researcher



June 4, 2025

# Contents

# 1 Introduction

## 1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

## 1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

## 1.3 Risk assessment

| Severity | Description |
|---|---|
| **Critical** | *Must* fix as soon as possible (if already deployed). |
| **High** | Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users. |
| **Medium** | Global losses <10% or losses to only a subset of users, but still unacceptable. |
| **Low** | Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies. |
| **Gas Optimization** | Suggestions around gas saving practices. |
| **Informational** | Suggestions around best practices or readability. |

### 1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

## 2 Security Review Summary

Virtuals Protocol is The Wall Street for AI Agents.

From May 22nd to May 24th the Cantina team conducted a review of virtual-protocol-contracts on commit hash d6491381. The team identified a total of **5** issues:

**Issues Found**

| Severity | Count | Fixed | Acknowledged |
|---|---|---|---|
| Critical Risk | 0 | 0 | 0 |
| High Risk | 0 | 0 | 0 |
| Medium Risk | 0 | 0 | 0 |
| Low Risk | 0 | 0 | 0 |
| Gas Optimizations | 2 | 0 | 0 |
| Informational | 3 | 0 | 0 |
| **Total** | **5** | **0** | **0** |

# 3 Findings

## 3.1 Gas Optimization

### 3.1.1 Gas savings in Genesis Contract Deployment

**Severity:** Gas Optimization

**Context:** GenesisLib.sol#L32

**Description:** When the `createGenesis` function is called, a new `Genesis` contract is deployed. However, the contract is currently not being cloned using a minimal proxy pattern, which would significantly reduce the gas cost associated with deployment.

Despite this, the `Genesis` contract is still made initializable. This forfeits the potential gas savings that could be achieved by marking many state variables as immutable since they do not change after initialization.

**Recommendation:** Either deploy a clone of the `Genesis` contract using a minimal proxy to reduce deployment costs, or initialize the state variables directly in the constructor. This will allow you to declare them as `immutable`, leading to lower gas usage when these variables are accessed later.

### 3.1.2 Redundant checks on allowance and balance

**Severity:** Gas Optimization

**Context:** AgentFactoryV5.sol#L168-L183, AgentFactoryV5.sol#L535-L550

**Description:** These checks are potentially redundant when `safeTransferFrom()` is used. Calls to `safeTransferFrom()` should revert when a call to `assetToken.transferFrom()` reverts or returns an ERC20 compliant value indicating the transfer failed.

**Recommendation:** If `assetToken` follows the ERC20 specification then these checks can be removed with the benefit of reducing gas cost and contract size.

## 3.2 Informational

### 3.2.1 Lack of documentation for externally callable functions

**Severity:** Informational

**Context:** *(No context files were provided by the reviewer)*

**Description:** Functions core to the functionality of the smart contract system do not have a company NatSpec documentation.

**Recommendation:** Introduce documentation for all functions that can be called externally. It's also worth considering adding similar documentation for other functions, as it helps improve maintainability and reduces cognitive load of the codebase.

Furthermore, it can be beneficial to revisit the structure of the code base and grouping of functions. Grouping similar (and/or externally callable) functions together can have tremendous impact on the time it takes to understand and change the contract.

Finally, introducing tests to execute the principal functionality of the contract will aid both as a form of documentation (tests demonstrate how the code is intended to function) and help discover unintended divergences in functionality between versions of the agent factory.

### 3.2.2 Use custom errors to reduce bytecode size

**Severity:** Informational

**Context:** Genesis.sol#L176-L194, AgentFactoryV5.sol#L92, AgentFactoryV5.sol#L168-L171, AgentFactoryV5.sol#L172-L176, AgentFactoryV5.sol#L353-L357, AgentFactoryV5.sol#L535-L544

**Description:** The codebase currently uses string-based revert statements to indicate errors. These strings increase the contract's bytecode size.

**Recommendation:** With the current version of Solidity, custom errors can be used with the require statement. Use them everywhere to reduce bytecode size.

### 3.2.3 Inheritance of non-upgradable contracts

**Severity:** Informational

**Context:** Genesis.sol#L13, AgentFactoryV5.sol#L22

**Description:** Upgradeable variants of contracts should be used when the contract is intended to be used in an upgradeable fashion. Currently the compiler will linearize the `AccessControl` constructor in the constructor of `AgentFactoryV5`. Note however that the impact is limited as the current version of AccessControlUpgradable does not have side effects.

Similarly `Genesis` inherits `AccessControlUpgradeable` but inherits non-upgradable `ReentrancyGuard`.

**Recommendation:** Use upgradable versions and apply appropriate initializers. For example, use `AccessControlUpgradeable` and initialize the contract in `AgentFactoryV5`. For `Genesis` it's also worth considering explicitly inheriting from Initializable. This contract is implicitly inherited from `AccessControlUpgradeable` and used directly. In such cases an explicit dependency is advisable.