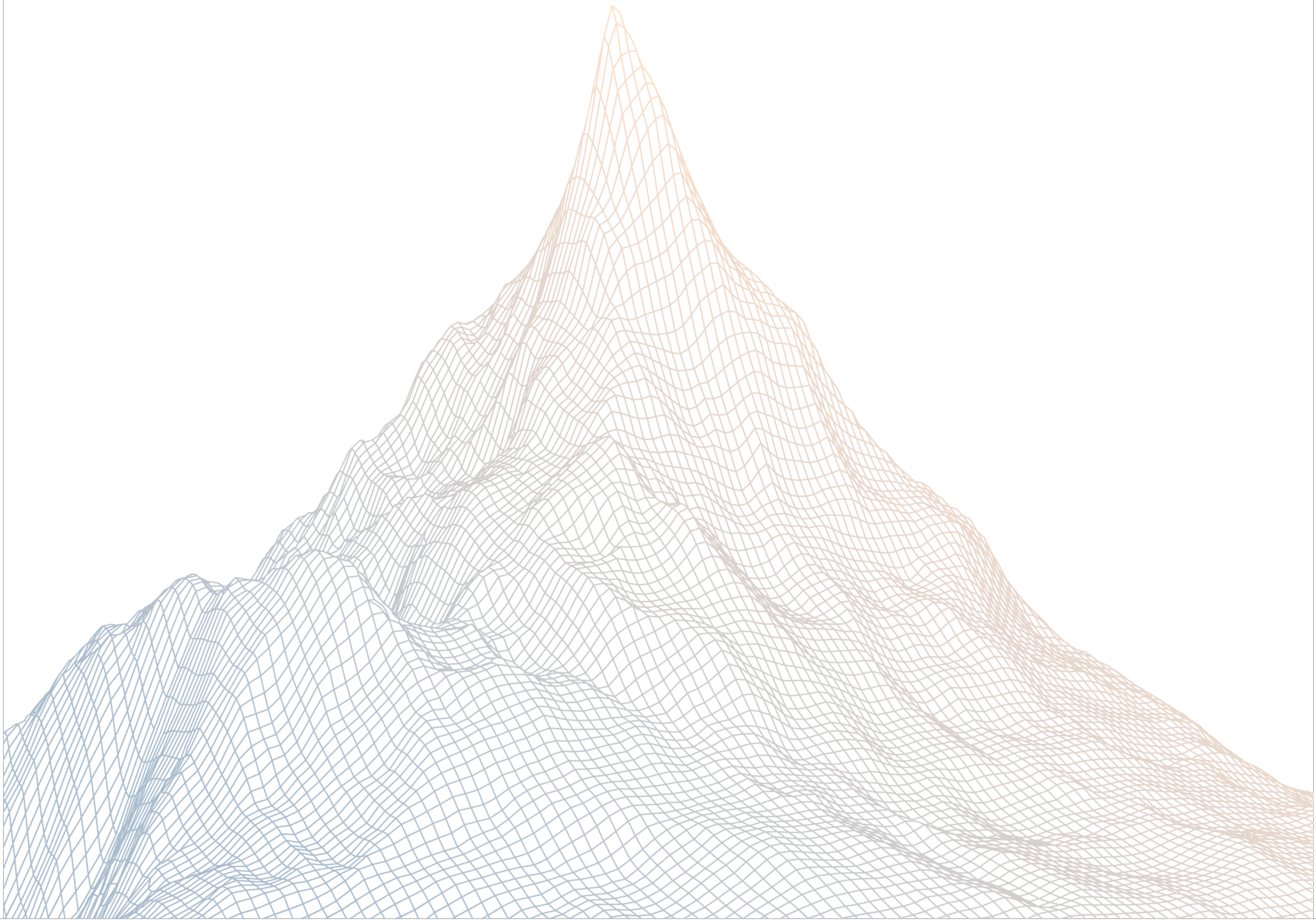
Zenith

Virtuals Protocol

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: April 4th to April 7th, 2025

AUDITED BY: Matte
SpicyMeatball

Contents

1 Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2 Executive Summary

2.1 About Virtuals Protocol

Virtuals Protocol is a society of productive AI agents, each designed to generate services or products and autonomously engage in commerce—either with humans or other agents—onchain. These agents are tokenized, represented by respective Agent Tokens, allowing for capital formation, permissionless participation, and aligned incentives among creators, investors, and agents.

The $VIRTUAL token is the base liquidity pair and transactional currency across all AI agent interactions, forming the monetary backbone of the ecosystem.

2.2 Scope

The engagement involved a review of the following targets:

Target	vp-genesis-contract
Repository	https://github.com/Virtual-Protocol/vp-genesis-contract
Commit Hash	`14b458cd8ffbb4a1f866cbc289f603afc0161365`
Files	`Genesis.sol` `FCGenesis.sol`

2.3 Audit Timeline

April 4, 2025	Audit start
April 7, 2025	Audit end
April 15, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	1
Medium Risk	1
Low Risk	3
Informational	0
Total Issues	**5**

3 Findings Summary

ID	Description	Status
H-1	Wrong creator address is set during Agent token launch	Resolved
M-1	Potential OOG error in onGenesisFailed	Resolved
L-1	Token refund inequity risk in Genesis contract's failure state transition	Resolved
L-2	Genesis failure after token launch may cause issues	Resolved
L-3	Additional checks needed when creating a Genesis	Resolved

4 Findings

4.1 High Risk

A total of 1 high risk findings were identified.

[H-1] Wrong creator address is set during Agent token launch

SEVERITY: High	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- Bonding.sol#L250
- AgentFactoryV3.sol#L555

Description:

When the Genesis contract initiates a token presale, the `Bonding.launch()` function sets `msg.sender`—which is the Genesis contract in this context—as the creator:

```
    function launch(
        string memory _name,
        string memory _ticker,
        uint8[] memory cores,
        string memory desc,
        string memory img,
        string[4] memory urls,
        uint256 purchaseAmount
    ) public nonReentrant returns (address, address, uint) {
        require(
            purchaseAmount > fee,
            "Purchase amount must be greater than fee"
        );

         --- SNIP ---

        Token memory tmpToken = Token({
>>          creator: msg.sender,
            token: address(token),
            agentToken: address(0),
            pair: _pair,
            data: _data,
            description: desc,
            cores: cores,
            image: img,
            twitter: urls[0],
            telegram: urls[1],
            youtube: urls[2],
            website: urls[3],
            trading: true, // Can only be traded once creator made initial
    purchase
            tradingOnUniswap: false
        });
        tokenInfo[address(token)] = tmpToken;
```

This `creator` field is later used during the final stage of the bonding curve to initiate Agent creation in the Agent Factory:

```
        function initFromBondingCurve(
            string memory name,
            string memory symbol,
            uint8[] memory cores,
            bytes32 tbaSalt,
            address tbaImplementation,
            uint32 daoVotingPeriod,
            uint256 daoThreshold,
            uint256 applicationThreshold_,
            address creator
        ) public whenNotPaused onlyRole(BONDING_ROLE) returns (uint256) {
            --- SNIP ---

            uint256 id = _nextId++;
            uint256 proposalEndBlock = block.number; // No longer required in v2
            Application memory application = Application(
                name,
                symbol,
                "",
                ApplicationStatus.Active,
                applicationThreshold_,
>>              creator,
                cores,
                proposalEndBlock,
                0,
                tbaSalt,
                tbaImplementation,
                daoVotingPeriod,
            daoThreshold
        );
```

Here, the creator is granted proposer privileges, which include:

- The ability to withdraw an application (if not yet finalized),
- Eligibility to become an LP staker in voting via veToken.

This is problematic, as the Genesis contract itself is not capable of interacting with the voting or Agent Factory contracts—making it an invalid creator.

Recommendations:

Pass the actual user who initiated the Genesis creation as the `creator` instead of using `msg.sender`. To implement this, modify the `Bonding.launch()` function to accept a creator address as an explicit parameter:

```
    function launch(
        string memory _name,
        string memory _ticker,
        uint8[] memory cores,
        string memory desc,
        string memory img,
        string[4] memory urls,
        uint256 purchaseAmount,
     address creator
    ) public nonReentrant returns (address, address, uint) {
        --- SNIP ---
        Token memory tmpToken = Token({
            creator: creator,
```

Virtuals: Resolved with @2f51fc9960..

Zenith: Verified. The creator is explicitly defined in `onGenesisSuccess` and subsequently passed as the proposer to AgentFactoryV3.

4.2 Medium Risk

A total of 1 medium risk findings were identified.

[M-1] Potential OOG error in onGenesisFailed

SEVERITY: Medium	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- Genesis.sol#L435-L447

Description:

The `Genesis` contract uses a dynamic array to store all users who participate in the genesis event by pledging their Virtual tokens:

```
    function participate(
        uint256 pointAmt,
        uint256 virtualsAmt,
        address userAddress
    )
        external
        nonReentrant
        whenNotCancelled
        whenNotFailed
        whenStarted
        whenNotEnded
    {
         --- SNIP ---

        // Update participant list
        if (mapAddrToVirtuals[userAddress] == 0) {
>>          participants.push(userAddress);
        }
```

In case the genesis event fails, the protocol will iterate over all elements in the array to refund the participants:

```
    function onGenesisFailed()
        external
        onlyRole(FACTORY_ROLE)
        nonReentrant
        whenNotCancelled
        whenNotFailed
        whenTokenNotLaunched
        whenEnded
    {
        isFailed = true;

        // Return all virtuals to participants
>>      for (uint256 i = 0; i < participants.length; i++) {
            address participant = participants[i];
            uint256 virtualsAmt = mapAddrToVirtuals[participant];
            if (virtualsAmt > 0) {
                // first clear the virtuals mapping of the user to prevent
    reentrancy attacks
                mapAddrToVirtuals[participant] = 0;
                // then transfer the virtuals
                IERC20(virtualTokenAddress).safeTransfer(
                    participant,
                    virtualsAmt
                );
                emit RefundClaimed(genesisId, participant, virtualsAmt);
            }
        }
```

This poses a risk because the `participants` array can theoretically grow to a size that could cause an out-of-gas error when iterating over it. A malicious actor could exploit this by creating multiple dust participation entries, bloating the list and potentially triggering this issue.

Recommendations:

Implement a reasonable limit on the number of participants allowed in a given genesis event.

Virtuals: Resolved with @2f51fc9960..

Zenith: Verified. `onGenesisFailed` now takes a list of participants as input, enabling a paginated-style refund process.

4.3 Low Risk

A total of 3 low risk findings were identified.

[L-1] Token refund inequity risk in Genesis contract's failure state transition

`SEVERITY`: Low	`IMPACT`: Low
`STATUS`: Resolved	`LIKELIHOOD`: Low

Target

- Genesis.sol

Description

When `onGenesisFailed()` function is called after the agent token has already been launched (i.e., `agentTokenAddress` is set), the contract attempts to refund all virtual tokens to participants based on their recorded contributions, but it does not account for the fact that some of these tokens have already been exhausted during the launch process.

The root cause of this issue is in the `onGenesisFailed()` function, which unconditionally attempts to refund the full amount of virtual tokens to all participants, regardless of whether the agent token has already been launched. When the agent token is launched via `onGenesisSuccess()`, a portion of the virtual tokens (specifically `reserveAmount`) is sent to the `virtualsFactoryAddress` to create the agent token. This means that after launch, the contract will not have enough virtual tokens to fully refund all participants if `onGenesisFailed()` is called.

The highest impact scenario occurs when:

1. The genesis is initially marked as successful via `onGenesisSuccess()`
2. The agent token is launched, consuming `reserveAmount` of virtual tokens
3. Later, due to some issue, the admin calls `onGenesisFailed()`
4. The contract attempts to refund all participants their full virtual token amounts
5. Some transfers will fail due to insufficient balance, or the contract will revert entirely

This creates a situation where some participants may receive full refunds while others receive nothing, depending on their position in the `participants` array, leading to an unfair distribution of remaining funds.

Recommendations

Prevent calling `onGenesisFailed()` after launch:

```
function onGenesisFailed() external onlyAdmin nonReentrant {
    require(!isFailed, "Genesis already failed");
    require(isEnded(), "Genesis not ended yet");
    require(agentTokenAddress == address(0), "Cannot fail after agent token
    launch");

    isFailed = true;

    // Rest of the function remains the same
}
```

Virtuals: Resolved with @3a9009c97ca...

Zenith: Verified.

[L-2] Genesis failure after token launch may cause issues

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- Genesis.sol#L284-L300

Description:

If the admin calls `onGenesisFailed()` after the token has already been launched (i.e., `agentTokenAddress` is set), there may be inconsistencies in refund amounts:

```
    function onGenesisFailed() external onlyAdmin nonReentrant {
        require(!isFailed, "Genesis already failed");
        require(isEnded(), "Genesis not ended yet");
        isFailed = true;

        // Return all virtuals to participants
        for (uint256 i = 0; i < participants.length; i++) {
            address participant = participants[i];
>>          uint256 virtualsAmt = mapAddrToVirtuals[participant];
            if (virtualsAmt > 0) {
                // first clear the virtuals mapping of the user to prevent
    reentrancy attacks
                mapAddrToVirtuals[participant] = 0;
                // then transfer the virtuals
                IERC20(virtualTokenAddress).transfer(participant,
    virtualsAmt);
                emit RefundClaimed(GENESIS_ID, participant, virtualsAmt);
            }
        }
```

While this function allows users to fully reclaim their virtual tokens, if the Agent token presale was already launched, some virtual tokens may have been consumed for the initial purchase. In such cases, there may not be enough remaining virtual tokens to fully refund all participants.

Recommendations:

The appropriate fix depends on whether calling `onGenesisFailed()` after a token launch is a valid use case (e.g., due to issues in the Agent Factory).

- If yes, implement a refund mechanism that accounts for virtual tokens already spent during the launch.
- If no, explicitly prevent the `onGenesisFailed()` function from executing once the Agent token has been deployed.

Virtuals: Resolved with @3a9009c97c...

Zenith: Verified. Once the token has been successfully launched, calling `onGenesisFailed` becomes impossible.

[L-3] Additional checks needed when creating a Genesis

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- FGenesis.sol#L151
- FGenesis.sol#L78
- Bonding.sol#L197-L202

Description:

Certain parameters used during Genesis creation can cause the deployment of the Agent token to revert:

```
function launch(
    string memory _name,
    string memory _ticker,
    uint8[] memory cores,
    string memory desc,
    string memory img,
    string[4] memory urls,
    uint256 purchaseAmount
) public nonReentrant returns (address, address, uint) {
    require(
        purchaseAmount > fee,
        "Purchase amount must be greater than fee"
    );

    require(cores.length > 0, "Cores must be provided");
```

In particular:

- `cores.length` must be greater than 0
- `reserveAmount` must be greater than fee

```
        if (isFirstLaunch) {
            // grant allowance to VirtualsFactory for launch
            IERC20(virtualTokenAddress).approve(virtualsFactoryAddress,
    reserveAmount);

            // Call launch function on VirtualsFactory
            (address funToken, , )
    = IVirtualsFactory(virtualsFactoryAddress).launch(
                genesisName,
                genesisTicker,
>>              genesisCores,
                genesisDesc,
                genesisImg,
                genesisUrls,
>>              reserveAmount
            );
```

If these conditions aren't met, Genesis will fail, and the admin will be required to initiate a full refund.

Recommendations:

Consider adding parameter validation to `FGenesis.reserveAmount` and `FGenesis.createGenesis` to ensure they meet the requirements of `Bonding.launch()`.

Virtuals: Resolved with @94f4112932...

Zenith: Verified. Additional validation has been implemented for the Genesis creation parameters.