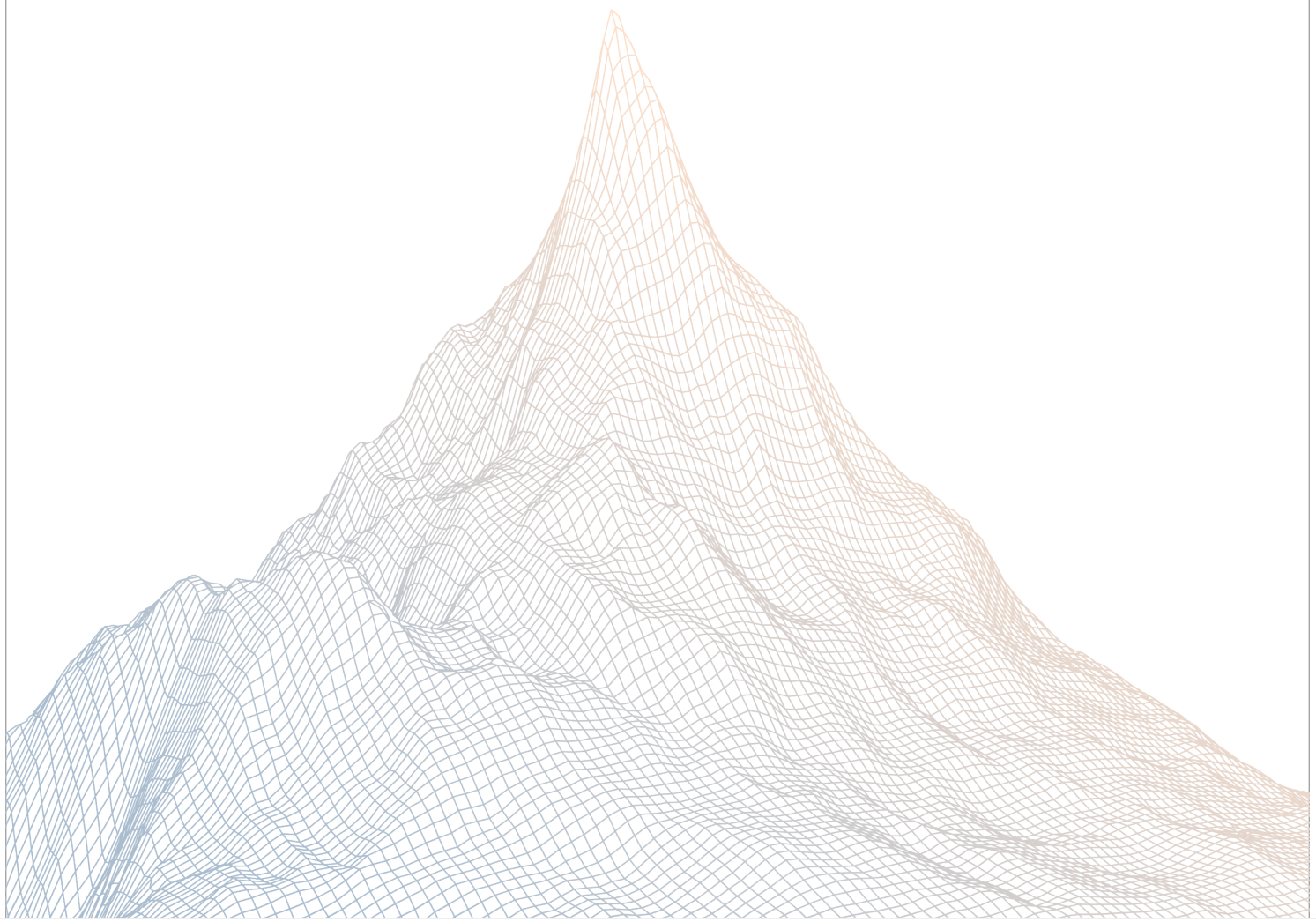
Zenith

Virtuals Protocol

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: May 2nd to May 5th, 2025

AUDITED BY: SpicyMeatball
zzykxx

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	protocol-contracts
Repository	https://github.com/Virtual-Protocol/protocol-contracts
Commit Hash	`8d55bfdcc7a573c4c1676602427f5012549417e8`
Files	`contracts/token/veVirtual.sol`

2.3 Audit Timeline

May 2, 2025	Audit start
May 3, 2025	Audit end
May 7, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	1
Medium Risk	3
Low Risk	3
Informational	0
Total Issues	**7**

3

Findings Summary

ID	Description	Status
H-1	Stakers with autoRenew enabled have an unfair advantage	Resolved
M-1	withdraw() switches indexes around which can lead to unintended operations when calling toggleAutoRenew(), extend() and withdraw() in a short time	Resolved
M-2	stake() allows to stake for 0 weeks leading to just-in-time voting	Resolved
M-3	Changing the maxWeeks variable can lead to unexpected consequences	Resolved
L-1	unbounded loops iterations on stakedAmountOf() and balanceOfAt()	Resolved
L-2	Suboptimal locks array management	Resolved
L-3	EIP712Upgradeable contract not initialized	Resolved

4

Findings

4.1 High Risk

A total of 1 high risk findings were identified.

[H-1] Stakers with `autoRenew` enabled have an unfair advantage

SEVERITY: High	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: High

Target

- veVirtual.sol#L162-L181

Description:

The `autoRenew` feature allows stakers to maintain maximum voting power over time without suffering from the usual vote decay:

```
    function _balanceOfLockAt(
        Lock memory lock,
        uint256 timestamp
    ) internal pure returns (uint256) {
        uint256 value = lock.value;
        if (lock.autoRenew) {
>>          return value;
        }
```

However, during staking, if `autoRenew` is enabled, the lock duration (`numWeeks`) is forcibly set to the maximum:

```
    function stake(
        uint256 amount,
        uint8 numWeeks,
        bool autoRenew
    ) external nonReentrant {
        require(amount > 0, "Amount must be greater than 0");
        require(numWeeks <= maxWeeks, "Num weeks must be less than max
    weeks");
```

```
        IERC20(baseToken).safeTransferFrom(_msgSender(), address(this),
    amount);

        if (autoRenew == true) {
>>          numWeeks = maxWeeks;
        }
```

As a result, users with `autoRenew` enabled enjoy full, non-decaying voting power while retaining the ability to withdraw after the standard `maxWeeks` lock period. In contrast, users who stake for `maxWeeks` without `autoRenew` must also wait the same withdrawal period, but their voting power decays over time.

Recommendations:

Restrict withdrawals for users with `autoRenew` enabled until they explicitly toggle it off. This would require them to wait the full maxWeeks period before being able to withdraw their stake.

Virtuals: Resolved with @8e8604a6a9...

Zenith: Verified.

4.2 Medium Risk

A total of 3 medium risk findings were identified.

[M-1] `withdraw()` switches indexes around which can lead to unintended operations when calling `toggleAutoRenew()`, `extend()` and `withdraw()` in a short time

`SEVERITY`: Medium	`IMPACT`: Medium
`STATUS`: Resolved	`LIKELIHOOD`: Medium

Target

- veVirtual.sol

Description:

The function withdraw() switches locks indexes.

This is problematic because functions such as toggleAutoRenew(), extend() and withdraw() itself accepts the index of a lock as input to know on which lock to perform operations. In case an user submits transactions regarding different locks in a short amount of time this can lead to failed transactions or performing operations on unintended locks. On top of this the order of operations is not guaranteed to match the order in which transactions were sent.

Recommendations:

When staking assing to each lock an incremental `ID` and use said `ID` as input in toggleAutoRenew(), extend() and withdraw() instead of the `index` in the array.

Virtuals: Resolved with @c241662f62...

Zenith: Verified. The issue has been fixed by adding a unique `id` to each lock and using the `id` as input parameter for `withdraw()`, `toggleAutoRenew()`, `extend()` and `getMaturity()`.

[M-2] `stake()` allows to stake for 0 weeks leading to just-in-time voting

`SEVERITY:` Medium	`IMPACT:` Medium
`STATUS:` Resolved	`LIKELIHOOD:` High

Target

- veVirtual.sol

Description:

The function stake() allows to pass `0` weeks as `numWeeks` input parameter.

Because the amount of votes is determined by the amount of tokens staked this allows the following scenario:

1. Alice stakes 100 tokens via `stake()` by passing `0` as `numWeeks`. This gives alice a power of 100.
2. Alice votes.
3. Alices withdraws via `withdraw()`, this is possible because Alice locked tokens for `0` weeks.

Recommendations:

Don't allow users to stake for `0` weeks. At the moment this implies the minimum staking amount is `1 week` but the `stake()` function can be changed to accept `seconds` or `days` as input instead of weeks, which allow for more granular staking time.

Virtuals: Resolved with @3d8c834bfd....

Zenith: Verified.

[M-3] Changing the `maxWeeks` variable can lead to unexpected consequences

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

Target

- veVirtual

Description:

Changing the `maxWeeks` parameter via setMaxWeeks() can lead to unintended consequences.

If the toggleAutoRenew() function is used to switch the autorenew to `off` the function sets the new end-time of the lock to:

```
lock.end = block.timestamp + uint(lock.numWeeks) * 1 weeks;
```

In this scenario `lock.numWeeks` is assumed to be equal to `maxWeeks` (as the toggle was `on`) but it contains the value `maxWeeks` had before it got changed via setMaxWeeks().

toggleAutoRenew() also sets the multiplier to:

```
uint multiplier = (uint(lock.numWeeks) * DENOM) / uint(maxWeeks);
```

In this scenario the multiplier might be higher than `100%` if `maxWeeks` got lowered and the specific lock `lock.numWeeks` is currently higher than `maxWeeks`.

In addition to this all locks have a `value` parameter that depend on `maxWeeks` and is not correctly updated to reflect the current value of `maxWeeks`. As an example the function _balanceOfLockAt() will return the `value` saved in storage for locks with `lock.autoRenew` set to `true` even if they should have an updated `value` as `maxWeeks` changed. This is also incoherent with the behaviour of the getMaturity() which returns the expiration time always based on the current value of `maxWeeks`.

Recommendations:

Fixing this requires:

- An update of the toggleAutoRenew() function to set `lock.end = block.timestamp + uint(maxWeeks) * 1 weeks;`
- An update of the toggleAutoRenew() function to lower `lock.numWeeks` to `maxWeeks` in case `lock.numWeeks > maxWeeks`
- An update to the contract to make sure the `value` of each lock is dependant on the current value of `maxWeeks`. A function can be used to determine the `value` instead of storing the `value` to storage.

Virtuals: Resolved with @bb47b4....

Zenith: Verified. The issue has been fixed by removing the `value` storage variable from the locks and by introducing an internal function `_calcValue()`. `toggleAutoRenew()` now correctly sets `lock.numWeeks` to `maxWeeks`. `_balanceOfLockAt()` correctly passes `maxWeeks` to `_calcValue()` when `lock.autoRenew` is set to `true`.

4.3 Low Risk

A total of 3 low risk findings were identified.

[L-1] unbounded loops iterations on `stakedAmountOf()` and `balanceOfAt()`

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

Target

- veVirtual.sol

Description:

The function stakedAmountOf() and balanceOfAt() both iterate on an unbounded array of locks of an account.

If a user has many locks it's possible for the functions to fail due to out-of-gas errors which can prevent delegations and can prevent getting the voting balance of an user.

Recommendations:

In the stake() function revert if the amount of locks the caller has is greater than a fixed, low amount.

Virtuals: Resolved with @370d7b0fb01....

Zenith: Verified.

[L-2] Suboptimal `locks` array management

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

Target

- veVirtual.sol#L176

Description:

In the `withdraw` function, the contract uses the `delete` keyword to remove a user's lock from the `locks` array:

```
    function withdraw(uint256 index) external nonReentrant {
        require(index < locks[_msgSender()].length, "Invalid index");
        Lock memory lock = locks[_msgSender()][index];
        require(
            block.timestamp >= lock.end || adminUnlocked,
            "Lock is not expired"
        );

        uint256 amount = lock.amount;

        uint256 lastIndex = locks[_msgSender()].length - 1;
        if (index ≠ lastIndex) {
            locks[_msgSender()][index] = locks[_msgSender()][lastIndex];
        }
>>      delete locks[_msgSender()][lastIndex];
```

While `delete` resets the contents at the specified index, it does not reduce the array length. As a result, the `locks` array can accumulate empty entries, leading to bloated storage. This causes increased gas usage in any operation that iterates through the array (e.g., `_getVotingUnits`) and could eventually result in out-of-gas errors for users with large lock histories.

Recommendations:

Consider using `pop` keyword.

Virtuals: Resolved with @0d8d19cc95...

Zenith: Verified.

[L-3] EIP712Upgradeable contract not initialized

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

Target

- EIP712Upgradeable.sol#L72

Description:

The `veVirtual` contract inherits from OpenZeppelin's `EIP712Upgradeable` as part of `VotesUpgradeable`. This inheritance is used to construct domain separators and hash messages for off-chain delegation via signatures. However, the `veVirtual` contract does not initialize the `EIP712Upgradeable` base during its setup, leaving the name and version storage variables unset. As a result, the generated domain separator may not match what external clients or tools would expect.

```
function __EIP712_init_unchained(string memory name, string memory version)
    internal onlyInitializing {
    EIP712Storage storage $ = _getEIP712Storage();
    $._name = name;
    $._version = version;

    // Reset prior values in storage if upgrading
    $._hashedName = 0;
    $._hashedVersion = 0;
}
```

Recommendations:

Call `__EIP712_init(name, version)` during the `veVirtual` contract initialization to ensure the domain separator and signature hashing behave correctly and predictably.

Virtuals: Resolved with @e65ca3da9e...

Zenith: Verified. EIP712 is initialized as expected.