Golom contest
Findings & Analysis Report

2023-04-05

Table of contents

- Overview
    - About C4
    - Wardens
- Summary
- Scope
- Severity Criteria
- High Risk Findings (11)
    - [H-01] Owner can not set the `ve` address via `RewardDistributor.addVoteEscrow`
    - [H-02] `VoteEscrowDelegation._writeCheckpoint` fails when `nCheckpoints` is 0
    - [H-03] GolomTrader's `_settleBalances` double counts protocol fee, reducing taker's payout for a NFT sold
    - [H-04] Old delegatee not deleted when delegating to new tokenId
    - [H-05] `addFee` will stop accumulating fee once `rewardToken` has reached max supply

    - [H-06] NFT transferring won’t work because of the external call to `removeDelegation`.
    - [H-07] `_writeCheckpoint` does not write to storage on same block
    - [H-08] Users can avoid paying fees while trading trustlessly & using Golom’s network effects
    - [H-09] Repeated calls to `multiStakerClaim` in the same block leads to loss of funds
    - [H-10] Upon changing of delegate, `VoteDelegation` updates both the previous and the current checkpoint
    - [H-11] Cannot remove delegation from a token to another token
- Medium Risk Findings (18)
    - [M-01] Use `call()` rather than `transfer()` on address payable
    - [M-02] Use `safeTransferFrom` Instead of `transferFrom` for ERC721
    - [M-03] Voter in `VoteEscrowCore` can permanently lock user tokens
    - [M-04] `VoteEscrowCore.safeTransferFrom` does not check correct magic bytes returned from receiver contract’s `onERC721Received` function
    - [M-05] Replay attack in case of hard fork
    - [M-06] Orders with `tokenAmt` of `type(uint256).max` cannot be cancelled by `GolomTrader.sol#cancelOrder`
    - [M-07] veNFT withdraw and merge fail for approved callers
    - [M-08] Pre-check is not correct
    - [M-09] `GolomToken.sol` doesn’t contain a function to mint treasury tokens
    - [M-10] Delegated NFTs that are withdrawn while still delegated will remain delegated even after burn
    - [M-11] When `MIN_VOTING_POWER_REQUIRED` is changed, previous votes are not affected
    - [M-12] Some setters’ timelock can be bypassed

- [M-13] Rewards owed burned NFT in `RewardDistributor.sol` become irretrievable
- [M-14] `VoteEscrowDelegation._transferFrom` can only be executed by the token owner
- [M-15] Griefer can minimize delegatee's voting power
- [M-16] `GolomTrader`: `validateOrder` function does not check if ecrecover return value is 0
- [M-17] NFTs that don't have a checkpoint can't be transferred
- [M-18] `fillAsk()` Allows for `msg.value` to be larger than require locking the excess in the contract

- Low Risk and Non-Critical Issues
  - Low Risk Issues
  - Non-critical Issues
  - L-1 Only a billion checkpoints available
  - L-02 Don't use `payable.transfer()` / `payable.send()`
  - L-03 Unused/empty `receive()` / `fallback()` function
  - L-04 `require()` should be used instead of `assert()`
  - L-05 Self-delegation is not automatic
  - L-06 Function may run out of gas
  - L-07 Vulnerable to cross-chain replay attacks due to static `DOMAIN_SEPARATOR` / `domainSeparator`
  - L-08 Wrong comment
  - L-09 Missing checks for `address(0x0)` when assigning values to `address` state variables
  - L-10 Missing event and or timelock for critical parameter change
  - L-11 Inconsistent spacing in comments
  - L-12 Typos
  - L-13 NatSpec is incomplete
  - N-01 Consider addings checks for signature malleability

    - N-02 `ecrecover()` signature validity not checked
    - N-03 Boilerplate not replaced
    - N-04 Remove commented out code
    - N-05 Invalid/extraneous/optional function definitions in interface
    - N-06 Remove `include` for hardhat's console
    - N-07 Contract implements interface without extending the interface
    - N-08 `require()` / `revert()` statements should have descriptive reason strings
    - N-09 `public` functions not called by the contract should be declared `external` instead
    - N-10 Non-assembly method available
    - N-11 `constant`s should be defined rather than using magic numbers
    - N-12 Numeric values having to do with time should use time units for readability
    - N-13 Large multiples of ten should use scientific notation (e.g. `1e6`) rather than decimal literals (e.g. `1000000`), for readability
    - N-14 Use a more recent version of solidity
    - N-15 Use scientific notation (e.g. `1e18`) rather than exponentiation (e.g. `10**18`)
    - N-16 Lines are too long
    - N-17 Inconsistent method of specifying a floating pragma
    - N-18 Variable names that consist of all capital letters should be reserved for `constant`/`immutable` variables
    - N-19 Non-library/interface files should use fixed compiler versions, not floating ones
    - N-20 File does not contain an SPDX Identifier
    - N-21 Event is missing `indexed` fields
    - N-22 Duplicated `require()`/`revert()` checks should be refactored to a modifier or function
- Gas Optimizations

- G-01 ++variable costs less gas than variable++, especially when it’s used in for-loops (—variable/variable— too)
- G-02 Access mappings directly rather than using accessor functions
- G-03 Cheaper input valdiations should come before expensive operations
- G-04 Help the optimizer by saving a storage variable’s reference instead of repeatedly fetching it
- G-05 The result of external function calls should be cached rather than re-calling the function
- G-06 Amounts should be checked for `0` before calling a transfer
- G-07 `public` functions to `external`
- G-08 Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas
- G-09 Default value initialization
- G-10 Multiplication/division by 2 should use bit shifting
- G-11 Splitting `require()` statements that use `&&` saves gas
- G-12 `abi.encode()` is less efficient than `abi.encodePacked()`
- G-13 `struct Order`: can be more tighly packed
- G-14 State variables can be packed into fewer storage slots
- G-15 `x += y` costs more gas than `x = x + y` for state variables
- G-16 `internal` and `private` functions only called once can be inlined to save gas
- G-17 Unchecked arithmetic
- G-18 Using `bool`s for storage incurs overhead
- G-19 Using `> 0` costs more gas than `!= 0` when used on a `uint` in a `require()` and `assert` statement
- G-20 Usage of `uints` / `ints` smaller than 32 bytes (256 bits) incurs overhead
- G-21 Use custom errors rather than `revert()` / `require()` strings to save gas

- G-22 Multiple `address` mappings can be combined into a single `mapping` of an `address` to a `struct`, where appropriate
- G-23 Functions guaranteed to revert when called by normal users can be marked `payable`
- G-24 Do not calculate constants
- G-25 `<array>.length` should not be looked up in every loop of a for-loop
- G-26 Copying struct to memory can be more expensive than just reading from storage
- G-27 `require()` / `revert()` strings longer than 32 bytes cost extra gas
- G-28 Remove or replace unused variables
- G-29 State variables only set in the constructor should be declared `immutable`
- G-30 State variables with values known at compile time should be constants
- G-31 State variables should be cached in stack variables rather than re-reading them from storage
- G-32 Using `private` rather than `public` for constants, saves gas
- G-33 Remove unreachable code
- G-34 No need to evaluate all expressions to know if one of them is true
- G-35 No need to read `tokenId` second time
- G-36 `last_point` value rewritten right after initialization
- G-37 Wasted gas on copying a struct
- G-38 Remove duplicate code
- G-39 No need for mapping `supportedInterfaces` to exist
- G-40 Same calculation twice
- G-41 Obsolete constants
- G-42 Variable `end` can be of type `uint128`

- Disclosures

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit contest outlined in this document, C4 conducted an analysis of the Golom smart contract system written in Solidity. The audit contest took place between July 26—August 1 2022.

Wardens

188 Wardens contributed reports to the Golom contest:

1. Krow10
2. kankodu
3. 0x52
4. IllIllI
5. scaraven
6. 0xsanson
7. cccz
8. 0xSky
9. kenzo
10. GimelSec (rayn and sces60107)
11. JohnSmith
12. 0xA5DF
13. berndartmueller
14. kyteg
15. GalloDaSballo

16. Bahurum
17. 0xpiglet
18. arcoun
19. MEP
20. rotcivegaf
21. GiveMeTestEther
22. CertoraInc (egjlmn1, OriDabush, ItayG, shakedwinder, and RoiEvenHaim)
23. zzzitron
24. rbserver
25. ElKu
26. TrungOre
27. 0x1f8b
28. kebabsec (okkothejawa and FlameHorizon)
29. carlitox477
30. 0xDjango
31. sseefried
32. Lambda
33. teddav
34. cryptphi
35. Dravee
36. simon135
37. obront
38. joestakey
39. hansfriese
40. kaden
41. panprog
42. ak1
43. async
44. reassor

45. hyh
46. RustyRabbit
47. minhquanym
48. DimitarDimitrov
49. auditor0517
50. 0xNineDec
51. csanuragjain
52. rokinot
53. djxploit
54. rajatbeladiya
55. Green
56. AuditsAreUS
57. shenwilly
58. Picodes
59. dipp
60. CRYP70
61. Treasure-Seeker
62. codexploder
63. horsefacts
64. chatch
65. MOndoHEHE
66. 0xHarry
67. jayjonah8
68. Bnke0x0
69. TomJ
70. benbaessler
71. JC
72. Deivitto
73. Rolezn

74. OxNazgul
75. cRat1st0s
76. Jmaxmanblue
77. Limbooo
78. wastewa
79. 0xf15ers (remora and twojoy)
80. 0x4non
81. __141345__
82. oyc_109
83. c3phas
84. 0xSmartContract
85. ajtra
86. MiloTruck
87. NoamYakov
88. Twpony
89. Ruhum
90. jayphbee
91. _Adam
92. brgltd
93. ellahi
94. Kenshin
95. RedOneN
96. saian
97. apostle0x01
98. Sm4rty
99. asutorufos
100. StyxRave
101. robee
102. Rohan16

103. sach1r0
104. 0xLovesleep
105. sashik_eth
106. 0xmatt
107. supernova
108. zuhaibmohd
109. delfin454000
110. fatherOfBlocks
111. Funen
112. jayfromthe13th
113. Junnon
114. lucacez
115. Maxime
116. mics
117. Aymen0909
118. Chinmay
119. gogo
120. m_Rassska
121. bin2chen
122. cryptonue
123. ych18
124. 8olidity
125. Chom
126. Jujic
127. Ch_301
128. Kumpa
129. Waze
130. 0xsolstars (Varun_Verma and masterchief)
131. cthulhu_cult (badbird and seanamani)

132. giovannidisiena
133. indijanc
134. StErMi
135. 0xackermann
136. 0xc0ffEE
137. saneryee
138. Soosh
139. Tadashi
140. codetilda
141. CryptoMartian
142. DevABDee
143. dirk_y
144. exd0tpy
145. Franfran
146. idkwhatimdoing
147. luckypanda
148. pedr02b2
149. Mohandes
150. neumo
151. erictee
152. durianSausage
153. ladboy233
154. Noah3o6
155. 0xKitsune
156. 0xsam
157. Chandr
158. CodingNameKiki
159. Fitraldys
160. gerdusx

161. Kaiziron
162. Migue
163. pfapostol
164. Randyyy
165. ReyAdmirado
166. rfa
167. samruna
168. tofunmi
169. Tomio
170. peritoflores
171. cloudjunky
172. i0001
173. PaludoX0
174. bardamu
175. bearonbike
176. bulej93
177. dharma09
178. immeas
179. navinavu

This contest was judged by LSDan.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 29 unique vulnerabilities. Of these vulnerabilities, 11 received a risk rating in the category of HIGH severity and 18 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 130 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 92 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Golom contest repository, and is composed of 6 smart contracts written in the Solidity programming language and includes 1,407 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities based on three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

For more information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website, specifically our section on Severity Categorization.

High Risk Findings (11)

[H-01] Owner can not set the `ve` address via `RewardDistributor.addVoteEscrow`

Submitted by berndartmueller, also found by 0x1f8b, 0x52, 0xA5DF, 0xsanson, auditor0517, CRYP70, GimelSec, hansfriese, hyh, Krow10, panprog, rajatbeladiya, rbserver, teddav, and TrungOre

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L300

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L173

On the initial `RewardDistributor.addVoteEscrow` call, the owner of the contract can set the `ve` address without a timelock (which is as intended according to the function documentation). However, as the function parameter `_voteEscrow` is not used for the assignment, instead the storage variable `pendingVoteEscrow` (which is not initialized, hence `address(0)`) is used, the `ve` storage variable can not be set to the provided `_voteEscrow` address.

This prevents setting the `ve` address (`ve` is set to `address(0)`) and therefore prevents `veNFT` holders to claim reward tokens and Ether rewards via `RewardDistributor.multiStakerClaim`.

Proof of Concept

RewardDistributor.sol#L300

```
function addVoteEscrow(address _voteEscrow) external onlyOwner {
    if (address(ve) == address(0)) {
        ve = VE(pendingVoteEscrow); // @audit-info The wrong var
    } else {
        voteEscrowEnableDate = block.timestamp + 1 days;
        pendingVoteEscrow = _voteEscrow;
    }
}
```

RewardDistributor.sol#L173

```
function multiStakerClaim(uint256[] memory tokenids, uint256[] m
    require(address(ve) != address(0), ' VE not added yet'); //

    ...
}
```

Recommended Mitigation Steps

Use the correct function parameter `_voteEscrow`:

```
    function addVoteEscrow(address _voteEscrow) external onlyOwner {
        if (address(ve) == address(0)) {
            ve = VE(_voteEscrow);
        } else {
            voteEscrowEnableDate = block.timestamp + 1 days;
            pendingVoteEscrow = _voteEscrow;
        }
    }
```

0xsaruman (Golom) confirmed

0xsaruman (Golom) resolved and commented:

> Resolved by removing the manually added timelocks and setting the Vote escrow in constructor and a function to change voteescrow by owner

> https://github.com/golom-protocol/contracts/commit/366c0455547041003c28f21b9afba48dc33dc5c7#diff-359fa403a6143105216e07c066e06ebb7ef2ba2d02f9d5465b042465d3f5bffbR297

[H-02] `VoteEscrowDelegation._writeCheckpoint` fails when `nCheckpoints` is 0

Submitted by GimelSec, also found by 0x52, 0xA5DF, 0xsanson, 0xSky, arcoun, Bahurum, berndartmueller, CertoraInc, cryptphi, ElKu, GalloDaSballo, hansfriese, JohnSmith, kenzo, kyteg, Lambda, MEP, panprog, rajatbeladiya, scaraven, simon135, Twpony, and zzzitron

https://github.com/code-423n4/2022-07-golom/blob/main/contracts/vote-escrow/VoteEscrowDelegation.sol#L101

https://github.com/code-423n4/2022-07-golom/blob/main/contracts/vote-escrow/VoteEscrowDelegation.sol#L82-L86

When a user call `VoteEscrowDelegation.delegate` to make a delegation, it calls `VoteEscrowDelegation._writeCheckpoint` to update the checkpoint of `toTokenId`. However, if `nCheckpoints` is 0, `_writeCheckpoint` always reverts. What's worse, `nCheckpoints` would be zero before any delegation has been made. In conclusion, users cannot make any delegation.

Proof of Concept

When a user call `VoteEscrowDelegation.delegate` to make a delegation, it calls `VoteEscrowDelegation._writeCheckpoint` to update the checkpoint of `toTokenId`.

https://github.com/code-423n4/2022-07-golom/blob/main/contracts/vote-escrow/VoteEscrowDelegation.sol#L82-L86

```
    function delegate(uint256 tokenId, uint256 toTokenId) extern
        require(ownerOf(tokenId) == msg.sender, 'VEDelegation: N
        require(this.balanceOfNFT(tokenId) >= MIN_VOTING_POWER_F

        delegates[tokenId] = toTokenId;
        uint256 nCheckpoints = numCheckpoints[toTokenId];

        if (nCheckpoints > 0) {
            Checkpoint storage checkpoint = checkpoints[toTokenI
            checkpoint.delegatedTokenIds.push(tokenId);
            _writeCheckpoint(toTokenId, nCheckpoints, checkpoint
        } else {
            uint256[] memory array = new uint256[](1);
            array[0] = tokenId;
            _writeCheckpoint(toTokenId, nCheckpoints, array);
        }

        emit DelegateChanged(tokenId, toTokenId, msg.sender);
    }
```

if `nCheckpoints` is 0, `_writeCheckpoint` always reverts.

Because `checkpoints[toTokenId][nCheckpoints - 1]` will trigger underflow in Solidity 0.8.11

https://github.com/code-423n4/2022-07-golom/blob/main/contracts/vote-escrow/VoteEscrowDelegation.sol#L101

```
    function _writeCheckpoint(
        uint256 toTokenId,
        uint256 nCheckpoints,
        uint256[] memory _delegatedTokenIds
    ) internal {
        require(_delegatedTokenIds.length < 500, 'VVDelegation:

        Checkpoint memory oldCheckpoint = checkpoints[toTokenId]
        …
    }
```

Recommended Mitigation Steps

Fix `_writeCheckpoint`

```
    function _writeCheckpoint(
        uint256 toTokenId,
        uint256 nCheckpoints,
        uint256[] memory _delegatedTokenIds
    ) internal {
        require(_delegatedTokenIds.length < 500, 'VVDelegation:



        if (nCheckpoints > 0 && oldCheckpoint.fromBlock == block
            Checkpoint memory oldCheckpoint = checkpoints[toToke
            oldCheckpoint.delegatedTokenIds = _delegatedTokenIds
        } else {
            checkpoints[toTokenId][nCheckpoints] = Checkpoint(bl
            numCheckpoints[toTokenId] = nCheckpoints + 1;
        }
    }
```

[zeroexdead (Golom) confirmed](#)

[zeroexdead (Golom) resolved and commented](#):

> Fixed. Ref: https://github.com/golom-protocol/contracts/commit/95e83a1abead683083b7ddf07853a26803c70b88

[H-03] GolomTrader's `_settleBalances` double counts protocol fee, reducing taker's payout for a NFT sold

Submitted by hyh, also found by 0x52, 0xSky, auditor0517, ElKu, kaden, Krow10, Lambda, Limbooo, obront, rbserver, rotcivegaf, RustyRabbit, scaraven, wastewa, and zzzitron

Currently `(o.totalAmt * 50) / 10000)` protocol fee share is multiplied by `amount` twice when being accounted for as a deduction from the total in amount due to the `msg.sender` taker calculations in _settleBalances(), which is called by fillBid() and fillCriteriaBid() to handle the payouts.

Setting the severity to be high as reduced payouts is a fund loss impact for taker, which receives less than it's due whenever `amount > 1`.

Notice that the amount lost to the taker is left on the contract balance and currently is subject to other vulnerabilities, i.e. can be easily stolen by an attacker that knowns these specifics and tracks contract state. When these issues be fixed this amount to be permanently frozen on the GolomTrader's balance as it's unaccounted for in all subsequent calculations (i.e. all the transfers are done with regard to the accounts recorded, this extra sum is unaccounted, there is no general native funds rescue function, so when all other mechanics be fixed the impact will be permanent freeze of the part of taker's funds).

Proof of Concept

_settleBalances() uses `(o.totalAmt - protocolfee - ...) * amount`, which is `o.totalAmt * amount - ((o.totalAmt * 50) / 10000) * amount * amount - ...`, counting protocol fee extra `amount - 1` times:

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L389-L399

```
        payEther(
            (o.totalAmt - protocolfee - o.exchange.paymentAm
                amount -
                p.paymentAmt,
            msg.sender
        );
    } else {
        payEther(
            (o.totalAmt - protocolfee - o.exchange.paymentAn
            msg.sender
        );
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L375-L400

```
    function _settleBalances(
        Order calldata o,
        uint256 amount,
        address referrer,
        Payment calldata p
    ) internal {
        uint256 protocolfee = ((o.totalAmt * 50) / 10000) * amou
        WETH.transferFrom(o.signer, address(this), o.totalAmt *
        WETH.withdraw(o.totalAmt * amount);
        payEther(protocolfee, address(distributor));
        payEther(o.exchange.paymentAmt * amount, o.exchange.paym
        payEther(o.prePayment.paymentAmt * amount, o.prePayment.
        if (o.refererrAmt > 0 && referrer != address(0)) {
            payEther(o.refererrAmt * amount, referrer);
            payEther(
                (o.totalAmt - protocolfee - o.exchange.paymentAn
                    amount -
                    p.paymentAmt,
                msg.sender
            );
        } else {
            payEther(
                (o.totalAmt - protocolfee - o.exchange.paymentAn
                msg.sender
            );
        }
```

Say, if `amount = 6`, while `((o.totalAmt * 50) / 10000) = 1 ETH`, `6 ETH` is total `protocolfee` and needs to be removed from `o.totalAmt * 6` to calculate taker's part, while `1 ETH * 6 * 6 = 36 ETH` is actually removed in the calculation, i.e. `36 - 6 = 30 ETH` of taker's funds will be frozen on the contract balance.

Recommended Mitigation Steps

Consider accounting for `amount` once, for example:

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L375-L403

```
        function _settleBalances(
            Order calldata o,
            uint256 amount,
            address referrer,
            Payment calldata p
        ) internal {
-           uint256 protocolfee = ((o.totalAmt * 50) / 10000) * amou
+           uint256 protocolfee = ((o.totalAmt * 50) / 10000);
            WETH.transferFrom(o.signer, address(this), o.totalAmt *
            WETH.withdraw(o.totalAmt * amount);
-           payEther(protocolfee, address(distributor));
+           payEther(protocolfee * amount, address(distributor));
            payEther(o.exchange.paymentAmt * amount, o.exchange.paym
            payEther(o.prePayment.paymentAmt * amount, o.prePayment.
            if (o.refererrAmt > 0 && referrer != address(0)) {
                payEther(o.refererrAmt * amount, referrer);
                payEther(
                    (o.totalAmt - protocolfee - o.exchange.paymentAm
                        amount -
                        p.paymentAmt,
                    msg.sender
                );
            } else {
                payEther(
                    (o.totalAmt - protocolfee - o.exchange.paymentAm
                    msg.sender
                );
            }
            payEther(p.paymentAmt, p.paymentAddress);
-           distributor.addFee([msg.sender, o.exchange.paymentAddres
+           distributor.addFee([msg.sender, o.exchange.paymentAddres
        }
```

[0xsaruman (Golom) confirmed]

[0xsaruman (Golom) resolved and commented]:

> Resolved [https://github.com/golom-protocol/contracts/commit/366c0455547041003c28f21b9afba48dc33dc5c7#diff-63895480b947c0761eff64ee21deb26847f597ebee3c024fb5aa3124ff78f6ccR390]

[H-04] Old delegatee not deleted when delegating to new tokenId

Submitted by Lambda, also found by 0x52, 0xA5DF, 0xDjango, 0xpiglet, 0xsanson, arcoun, Bahurum, berndartmueller, cccz, dipp, GalloDaSballo, GimelSec, GiveMeTestEther, Green, kenzo, kyteg, MEP, neumo, obront, panprog, rajatbeladiya, scaraven, and Twpony

[VoteEscrowDelegation.sol#L80]

In `delegate`, when a user delegates to a new tokenId, the tokenId is not removed from the current delegatee. Therefore, one user can easily multiply his voting power, which makes the toking useless for voting / governance decisions.

Proof Of Concept

Bob owns the token with ID 1 with a current balance of 1000. He also owns tokens 2, 3, 4, 5. Therefore, he calls `delegate(1, 2)`, `delegate(1, 3)`, `delegate(1, 4)`, `delegate(1, 5)`. Now, if there is a governance decision and `getVotes` is called, Bobs balance of 1000 is included in token 2, 3, 4, and 5. Therefore, he quadrupled the voting power of token 1.

Recommended Mitigation Steps

Remove the entry in `delegatedTokenIds` of the old delegatee or simply call `removeDelegation` first.

[zeroexdead (Golom) confirmed]

[zeroexdead (Golom) commented]:

> Fixed.

> Ref: https://github.com/golom-protocol/contracts/commit/c74d95b4105eeb878d2781982178db5ca08a1a9b

[H-05] `addFee` will stop accumulating fee once `rewardToken` has reached max supply

Submitted by shenwilly, also found by 0x52, berndartmueller, GimelSec, GiveMeTestEther, kaden, Lambda, MOndoHEHE, obront, Picodes, rbserver, reassor, rokinot, and scaraven

`RewardDistributor` will stop accumulating fees for staker rewards once `rewardToken` supply has reached the maximum supply (1 billion).

Vulnerability Details

[RewardDistributor.sol#L98-L138]

```
    function addFee(address[2] memory addr, uint256 fee) public onl
        if (rewardToken.totalSupply() > 1000000000 * 10**18) {
            // if supply is greater then a billion dont mint anythir
            return;
        }

        ...

        feesTrader[addr[0]][epoch] = feesTrader[addr[0]][epoch] + fe
        feesExchange[addr[1]][epoch] = feesExchange[addr[1]][epoch]
        epochTotalFee[epoch] = epochTotalFee[epoch] + fee;
    }
```

The check at the beginning of `addFee` is supposed to stop `RewardDistributor` from minting additional rewardToken once it has reached 1 billion supply. However, the current implementation has a side effect of causing the function to skip recording accumulated trading fees (the last 3 lines of the function). This will cause stakers to lose their trading fee rewards once the max supply has been reached, and the funds will be permanently locked in the contract.

Proof of Concept

- Alice staked `GOLOM` to receive fee rewards from `RewardDistributor`.
- `GOLOM` supply reaches 1 billion token.
- Traders keep trading on `GolomTrader`, sending protocol fees to `RewardDistributor`. However, `RewardDistributor.addFee` does not update the fee accounting.
- Alice won't receive any fee reward and protocol fees are stuck in the contract.

Recommended Mitigation Steps

Modify `addFee` so that the check won't skip accruing trade fees:

```
    function addFee(address[2] memory addr, uint256 fee) public only
        if (block.timestamp > startTime + (epoch) * secsInDay) {
            uint256 previousEpochFee = epochTotalFee[epoch];
            epoch = epoch + 1;

            if (rewardToken.totalSupply() > 1000000000 * 10**18) {
                emit NewEpoch(epoch, 0, 0, previousEpochFee);
            } else {
                uint256 tokenToEmit = (dailyEmission * (rewardToken.
                    rewardToken.totalSupply();
                uint256 stakerReward = (tokenToEmit * rewardToken.ba

                rewardStaker[epoch] = stakerReward;
                rewardTrader[epoch] = ((tokenToEmit - stakerReward)
                rewardExchange[epoch] = ((tokenToEmit - stakerRewar
                rewardToken.mint(address(this), tokenToEmit);
                epochBeginTime[epoch] = block.number;
                if (previousEpochFee > 0) {
                    if (epoch == 1){
                        epochTotalFee[0] =  address(this).balance; /
                        weth.deposit{value: address(this).balance}()
                    }else{
                        weth.deposit{value: previousEpochFee}();
                    }
                }
                emit NewEpoch(epoch, tokenToEmit, stakerReward, prev
            }
        }
        feesTrader[addr[0]][epoch] = feesTrader[addr[0]][epoch] + fe
        feesExchange[addr[1]][epoch] = feesExchange[addr[1]][epoch]
        epochTotalFee[epoch] = epochTotalFee[epoch] + fee;
        return;
    }
```

0xsaruman (Golom) confirmed

0xsaruman (Golom) resolved and commented:

> Resolved in https://github.com/golom-protocol/contracts/commit/192e152dde2eed6c01a3945aa5fd223ff786ca5e

[H-06] NFT transferring won't work because of the external call to `removeDelegation`.

Submitted by CertoraInc, also found by 0xA5DF, 0xsanson, Bahurum, carlitox477, cryptphi, GalloDaSballo, kenzo, MEP, and TrungOre

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowDelegation.sol#L242

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowDelegation.sol#L211

The `VoteEscrowDelegation._transferFrom` function won't work because it calls `this.removeDelegation(_tokenId)`. The `removeDelegation` function is external, so when the call is done by `this.removeDelegation(_tokenId)` msg.sender changes to the contract address.

This causes the check in the `function to (most likely) fail because the contract is not the owner of the NFT, and that will make the function revert.

```
require(ownerOf(tokenId) == msg.sender, 'VEDelegation: Not allowed');
```

Recommended Mitigation Steps

Make the `removeDelegation` function public and call it without changing the context (i.e. without changing msg.sender to the contract's address).

[zeroexdead (Golom) confirmed](#)

[zeroexdead (Golom) commented](#):

> Fixed.

> Ref: https://github.com/golom-protocol/contracts/commit/10ec920765a5ee2afc2fe269d32ea9138d1156b6

[0xsaruman (Golom) resolved](#)

[H-07] `_writeCheckpoint` does not write to storage on same block

Submitted by async, also found by 0xA5DF, 0xpiglet, 0xsanson, ak1, DimitarDimitrov, Dravee, ElKu, IllIllI, JohnSmith, kenzo, and scaraven

[VoteEscrowDelegation.sol#L101-L108](#)

In `VoteEscrowDelegation._writeCheckpoint`, when the checkpoint is overwritten in the same block the new value is set with `memory oldCheckpoint` and thus is never written to storage.

```
    Checkpoint memory oldCheckpoint = checkpoints[toTokenId][nCheckp

    if (nCheckpoints > 0 && oldCheckpoint.fromBlock == block.number)

    oldCheckpoint.delegatedTokenIds = _delegatedTokenIds;
    }
```

Users that remove and delegate a token (or call `delegate` on the same token twice) in the same block will only have their first delegation persisted.

Proof of Concept

- User delegates a `tokenId` by calling `delegate`.
- In the same block, the user decides to delgate the same token to a different token ID and calls `delegate` again which calls `_writeCheckpoint`. Since this is the second transaction in the same block the if statement in the code block above executes and stores `_delegatedTokenIds` in `memory oldCheckpoint`, thus not persisting the array of `_delegatedTokenIds` in the checkpoint.

Recommended Mitigation Steps

Define the `oldCheckpoint` variable as a `storage` pointer:

```
 Checkpoint storage oldCheckpoint = checkpoints[toTokenId]
[nCheckpoints - 1];
```

0xA5DF (warden) commented:

> Just want to add to the impact (in case the judges consider to decrease severity), in my report of this bug (#625) I've mentioned a more severe impact:

> An attacker can use this to multiplying his delegation power endlessly, by adding a delegation and removing it in the same block (using a contract to run those 2 functions in the same tx). The delegation will succeed but the removal will fail, this way each time this runs the user delegates again the same token.

zeroexdead (Golom) confirmed

zeroexdead (Golom) commented:

> Fixed. Ref: https://github.com/golom-protocol/contracts/commit/74b2e718f6ae9da815b52242a44451527d60d1ae

0xsaruman (Golom) resolved

[H-08] Users can avoid paying fees while trading trustlessly & using Golom's network effects

Submitted by kankodu

- If a maker makes below mentioned `AvoidsFeesContract` a reservedAddress and hides the info about how much they want their NFT in order.root, they can avoid paying fees while trading trustlessly and using the nework effects of golom maketplace with 0 o.totalAmt. See POC to get a better idea.
- Here the maker uses order.root to hide the amount they want to get paid because it is much cleaner for a POC.
    - But since golom does not have an API where user can submit a signature without using the frontend, they will use something like deadline to hide the amount they want to get paid.
    - Reason they would use deadline is because that is something they can control in the golom NFT frontend
    - They can pack the information about deadline and amount they want to get paid, in one uint256 as a deadline and then the check in the contract would look a different

Proof of Concept

- Clone the repo and run `yarn`
- Create a `AvoidsFeesContract.sol` contract in `contracts/test/` folder with following code

```
//contract that avoids paying fees everytime

pragma solidity 0.8.11;

import "../core/GolomTrader.sol";

//A maker will be gurranteed a payout if it makes this contract
//Users will use this every time to trade to avoid paying fees
```

```
//They use the networking effects of the golom marketplace withc
contract AvoidsFeesContract {
    GolomTrader public immutable golomTrader;

    constructor(GolomTrader _golomTrader) {
        golomTrader = _golomTrader;
    }

    function fillAsk(
        GolomTrader.Order calldata o,
        uint256 amount,
        address referrer,
        GolomTrader.Payment calldata p,
        address receiver
    ) public payable {
        require(
            o.reservedAddress == address(this),
            "not allowed if signer has not reserved this contrac
        ); //the signer will only allow this contract to execute
        require(
            p.paymentAddress == o.signer,
            "signer needs to be the payment address"
        );
        //I am using root as an example because it is much clear
        //but since golom does not have an API where user can su
        //Reason they would use deadline is because that is some
        //They can pack the information about deadline and amour
        require(
            p.paymentAmt == uint256(o.root),
            "you need to pay what signer wants"
        ); //the maker will hide the payment info in oder.root

        golomTrader.fillAsk{value: msg.value}(
            o,
            amount,
            referrer,
            p,
            receiver = msg.sender
        );
    }
}
```

- Add following test in `test/GolomTrader.specs.ts` here.

- Also, add `const AvoidsFeesContractArtifacts = ethers.getContractFactory('AvoidsFeesContract');` after this line and `import { AvoidsFeesContract as AvoidsFeesContractTypes } from '../typechain/AvoidsFeesContract';` after this line.
- Run `npx hardhat compile && npx hardhat test`

```
    it.only('should allow malicious contract to execute the t
        //deploy the malicious contract
        const avoidsFeesContract: AvoidsFeesContractTypes = 

        //here the frontend calculates exchangeAmount and pr
        //as far as the frontend is concerned, the maker inp
        let exchangeAmount = ethers.utils.parseEther('0'); /
        let prePaymentAmt = ethers.utils.parseEther('0'); //
        let totalAmt = ethers.utils.parseEther('0');
        let tokenId = await testErc721.current();

        let nftValueThatMakerWants = ethers.utils.parseEther

        const order = {
            collection: testErc721.address,
            tokenId: tokenId,
            signer: await maker.getAddress(),
            orderType: 0,
            totalAmt: totalAmt,
            exchange: { paymentAmt: exchangeAmount, paymentA
            prePayment: { paymentAmt: prePaymentAmt, payment
            isERC721: true,
            tokenAmt: 1,
            refererrAmt: 0,
            root: ethers.utils.hexZeroPad(nftValueThatMakerW
            reservedAddress: avoidsFeesContract.address,
            nonce: 0,
            deadline: Date.now() + 100000,
            r: '',
            s: '',
            v: 0,
        };

        let signature = (await maker._signTypedData(domain, 

        order.r = '0x' + signature.substring(0, 64);
        order.s = '0x' + signature.substring(64, 128);
        order.v = parseInt(signature.substring(128, 130), 16

        let makerBalanceBefore = await ethers.provider.getBa

        await avoidsFeesContract.connect(taker).fillAsk(
            order,
            1,
            '0x0000000000000000000000000000000000000000',
            {
                paymentAmt: nftValueThatMakerWants,
                paymentAddress: order.signer,
            },
            receiver,
            {
                value: nftValueThatMakerWants,
            }
        );

        let makerBalanceAfter = await ethers.provider.getBal

        expect(await testErc721.balanceOf(await taker.getAdd
        expect(makerBalanceAfter.sub(makerBalanceBefore)).to

    });
```

Tools Used

- The repo itself. (hardhat)

Recommended Mitigation Steps

- Make sure that o.totalAmt is greater than p.paymentAmt in addition to this check

0xsaruman (Golom) confirmed

0xsaruman (Golom) resolved and commented:

> Circumvented by putting this line in the code
>
> ```
> require(o.totalAmt * amount * 15/100 >= p.paymentAmt, 'can only pay
> 15% extra');
> ```

[H-09] Repeated calls to `multiStakerClaim` in the same block leads to loss of funds

Submitted by Krow10

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L172-L210

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L185

A malicious user can repeatedly claim the same staker reward for an epoch, provided the transactions all happen in the same block. This can effectively be done using services like Flashbots bundles and will result in the draining of the WETH balance of the `RewardDistributor` contract.

The idea is to bypass the require statement line 185 which checks if a claim has been already done for the epoch, **for a specific token ID**. By moving the locked tokens in a new lock, a new token ID will be generated and can be used to claim the rewards again, **if the transaction happens in the same block for which the epoch is updated.**

Indeed, when `multiStakerClaim()` is called, the `rewardETH` will be calculated from the amount of tokens locked in `tokenids[tindex]` at the block that triggered the epoch change (variable `epochBeginTime`). If, during this time, an attacker transfers its staked tokens to a new vault using the `merge` function of the VE token, the function will calculate the amount of staked tokens for the newly created tokenID **as the same as the original tokenID reward.**

A example abuse will look like this (pseudo-code adapted from the PoC) :

```
lockID = voteEscrow.create_lock(amount, 1 week); // Create lock
// IN THE BLOCK OF EPOCH CHANGE
rewardDistributor.multiStakerClaim([lockId], [0]); // Claim epoc
voteEscrow.create_lock(1, 1 week); // Create lock #2 (requires 1
voteEscrow.merge(lockId, lockId + 1); // Transfer lock #1 tokens
```

```
    rewardDistributor.multiStakerClaim([lockId + 1], [0]); // Claim
    // repeat ...
```

To abuse this, the attacker needs to follow this steps:

- Have some locked Golom tokens.
- Wait for a `addFee` call that will trigger an epoch change (this can be monitored by looking at the mempool or predicted from block timestamps). Services like Flashbots also allows for specifying a range of blocks for bundles for better targeting.
- Send a bundle of transactions to be included with the block containing the epoch changing transaction (see the PoC for an example of transactions).

Note that this needs to succeed only once to allow an attacker to drain all WETH funds so if the bundle isn't included for a particular epoch, given the frequency of epoch changes, the bundle will eventually be included and trigger the exploit.

Proof of Concept

See warden's original submission for full proof of concept.

Recommended Mitigation Steps

I initially thought about a few possible solutions:

- Checking a lock creation time to prevent claiming from locks created in the same block **but the attacker can just create the blocks beforehand.**
- Tracking the `msg.sender` or `tx.origin` for preventing multiple calls to `multiStakerClaim` in the same block **but the attacker can just send transactions from different addresses.**
- Preventing the merging of locks **but the attacker can just create locks in advance and withdraw/add funds continuously between old/new locks.**

None really fixes the vulnerability as it comes from the feature of **locks being tradable** meaning it's not practically feasable to know if a lock has already be claimed by an individual **just by looking at the lock ID.**

A possible solution would be to find a way to prevent multiple calls to the same function within a block or better, make a checkpoint of the locks balances for each `epochBeginTime` and uses these values for calculating the rewards (instead of querying the VE contract in the loop).

[0xsaruman (Golom) confirmed]

[0xsaruman (Golom) resolved and commented]:

> Removed `merge()`
> Ref: https://github.com/golom-protocol/contracts/commit/b987077f2a227273bc7051e382bd55264162a77e

[H-10] Upon changing of delegate, `VoteDelegation` updates both the previous and the current checkpoint

Submitted by kenzo, also found by 0xA5DF, 0xpiglet, 0xsanson, arcoun, Bahurum, and IllIllI

https://github.com/code-423n4/2022-07-golom/blob/main/contracts/vote-escrow/VoteEscrowDelegation.sol#L79

https://github.com/code-423n4/2022-07-golom/blob/main/contracts/vote-escrow/VoteEscrowDelegation.sol#L213

The contract is accidently editing both the previous and current checkpoint when changing/removing a delegate.

Impact

Incorrect counting of votes.

Proof of Concept

If in `delegate` the delegate already has checkpoints, the function will grab the latest checkpoint, and add the `tokenId` to it. Note that it changes the storage variable.

```
            if (nCheckpoints > 0) {
                Checkpoint storage checkpoint = checkpoints[toTokenI
                checkpoint.delegatedTokenIds.push(tokenId);
                _writeCheckpoint(toTokenId, nCheckpoints, checkpoint
```

It then calls `_writeCheckpoint`, which [will add](#) a new checkpoint if there’s no checkpoint created for this block yet:

```
            Checkpoint memory oldCheckpoint = checkpoints[toTokenId]

            if (nCheckpoints > 0 && oldCheckpoint.fromBlock == block
                oldCheckpoint.delegatedTokenIds = _delegatedTokenIds
            } else {
                checkpoints[toTokenId][nCheckpoints] = Checkpoint(bl
                numCheckpoints[toTokenId] = nCheckpoints + 1;
            }
```

Therefore, if this function has created a new checkpoint with the passed `_delegatedTokenIds`, we already saw that the previous function has already added `tokenId` to the previous checkpoint, so now both the new checkpoint and the previous checkpoint will have `tokenId` in them.
This is wrong as it updates an earlier checkpoint with the latest change.

The same situation happens in [`removeDelegation`](#).

Recommended Mitigation Steps

When reading the latest checkpoint:

```
    Checkpoint storage checkpoint = checkpoints[toTokenId][nCheckpoi
```

Change the `storage` to `memory`. This way it will not affect the previous checkpoint, but will pass the correct updated array to `_writeCheckpoint`, which will then write/update the correct checkpoint.

[**zeroexdead (Golom) confirmed and commented**](#):

> Fixed `delegate()` : https://github.com/golom-protocol/contracts/commit/8a8c89beea22cd57f4ffaf3d0defcce863e9657f

> Fixed `removeDelegation()` : https://github.com/golom-protocol/contracts/commit/72350b0a3bdae4f21e2f015327037080f6bab867

[LSDan (judge) increased severity to High and commented]:

> I went back and forth on if this was a duplicate of [H-04 (#169)] or not. The two issues are so similar it's hard to pull them apart. Ultimately I do see the difference, mainly that this version of the issue results in a retroactive manipulation of voting power whereas the other issue allows the creation of infinite voting power. I'm upgrading this to high risk because it effectively destroys the integrity of the voting system which impacts every aspect of the protocol which is subject to vote.

[H-11] Cannot remove delegation from a token to another token

Submitted by Bahurum, also found by 0x52, 0xA5DF, 0xsanson, berndartmueller, cccz, CertoraInc, dipp, GalloDaSballo, GimelSec, Green, IllIllI, kenzo, MEP, panprog, and scaraven

[VoteEscrowDelegation.sol#L213]

A user who has delegated the vote of a veGolom token (that he/she owns) to another veGolom token cannot remove the delegation, so the delegatee token will permanently hold the voting power of the delegator token.

Proof of Concept

A user tries to remove the delegation from `tokenId` he/she owns to the delegated token, calling `removeDelegation(uint256 tokenId)`.
The delegation should be removed at the lines:

```
        Checkpoint storage checkpoint = checkpoints[tokenId][nCh
        removeElement(checkpoint.delegatedTokenIds, tokenId);
```

but the array `checkpoint.delegatedTokenIds` is the list of **delegators** to `tokenId` **itself**. So, unless the delegation was from the token to itself, `removeDelegation` does nothing.

Recommended Mitigation Steps

Two fixes are proposed:

1. Add the delegatee as an argument to `removeDelegation` and remove `tokenId` from its list of delegators:

```
-   function removeDelegation(uint256 tokenId) external {
+   function removeDelegation(uint256 tokenId, uint256 toTokenIc
        require(ownerOf(tokenId) == msg.sender, 'VEDelegation: N
        uint256 nCheckpoints = numCheckpoints[tokenId];
-       Checkpoint storage checkpoint = checkpoints[tokenId][nCh
+       Checkpoint storage checkpoint = checkpoints[toTokenId][r
        removeElement(checkpoint.delegatedTokenIds, tokenId);
        _writeCheckpoint(tokenId, nCheckpoints, checkpoint.deleg
    }
```

or

2. Load the delegatee from the mapping `delegates` which maps each delegator to its current delegatee:

```
    function removeDelegation(uint256 tokenId) external {
        require(ownerOf(tokenId) == msg.sender, 'VEDelegation: N
+       uint256 toTokenId = delegates[tokenId];
        uint256 nCheckpoints = numCheckpoints[tokenId];
-       Checkpoint storage checkpoint = checkpoints[tokenId][nCh
+       Checkpoint storage checkpoint = checkpoints[toTokenId][r
        removeElement(checkpoint.delegatedTokenIds, tokenId);
        _writeCheckpoint(tokenId, nCheckpoints, checkpoint.deleg
    }
```

kenzo (warden) commented:

> Note that in the mitigation, `nCheckpoints` should access `toTokenId` instead of `tokenId`.

zeroexdead (Golom) confirmed

zeroexdead (Golom) commented:

> Fixed. https://github.com/golom-protocol/contracts/commit/4b19fce83ad53bc56b1bad058e1e88d90acda444

0xsaruman (Golom) resolved

LSDan (judge) increased severity to High and commented:

> I agree with the other wardens who rated this high risk. It has a direct impact on the functioning of the protocol and allows for a myriad of governance attacks.

Medium Risk Findings (18)

[M-01] Use `call()` rather than `transfer()` on address payable

Submitted by cloudjunky, also found by __141345__, _Adam, 0x1f8b, 0x4non, 0x52, 0xDjango, 0xf15ers, 0xHarry, 0xNazgul, 0xNineDec, 0xsanson, 0xsolstars, 8olidity, arcoun, asutorufos, bardamu, bearonbike, bin2chen, Bnke0x0, brgltd, bulej93, c3phas, carlitox477, cccz, CertoraInc, Chom, codexploder, cRat1st0s, cryptonue, cryptphi, cthulhu_cult, Deivitto, dharma09, dipp, djxploit, Dravee, durianSausage, ellahi, GalloDaSballo, GimelSec, giovannidisiena, hansfriese, horsefacts, hyh, IllIllI, immeas, indijanc, jayjonah8, jayphbee, Jmaxmanblue, joestakey, JohnSmith, Jujic, Kenshin, kenzo, Krow10, kyteg, ladboy233, Lambda, MEP, minhquanym, navinavu, Noah3o6, obront, oyc_109, peritoflores, rbserver, reassor, RedOneN, rokinot, rotcivegaf, Ruhum, saian, scaraven, shenwilly, simon135, sseefried, StErMi, StyxRave, teddav, TomJ, Treasure-Seeker, TrungOre, and zzzitron

L154 in GolomTrader.sol uses `.transfer()` to send ether to other addresses. There are a number of issues with using `.transfer()`, as it can fail for a number of reasons (specified in the Proof of Concept).

Proof of Concept

1. The destination is a smart contract that doesn't implement a `payable` function or it implements a `payable` function but that function uses more than 2300 gas units.
2. The destination is a smart contract that doesn't implement a `payable fallback` function or it implements a `payable fallback` function but that function uses more than 2300 gas units.
3. The destination is a smart contract but that smart contract is called via an intermediate proxy contract increasing the case requirements to more than 2300 gas units. A further example of unknown destination complexity is that of a multisig wallet that as part of its operation uses more than 2300 gas units.
4. Future changes or forks in Ethereum result in higher gas fees than transfer provides. The `.transfer()` creates a hard dependency on 2300 gas units being appropriate now and into the future.

Tools Used

Vim

Recommended Remediation Steps

Instead use the `.call()` function to transfer ether and avoid some of the limitations of `.transfer()`. This would be accomplished by changing `payEther()` to something like;

```
    (bool success, ) = payable(payAddress).call{value: payAmt}("");
    require(success, "Transfer failed.");
```

Gas units can also be passed to the `.call()` function as a variable to accomodate any uses edge cases. Gas could be a mutable state variable that can be set by the contract owner.

Oxsaruman (Golom) confirmed

[0xsaruman (Golom) resolved and commented](#):

> Resolved https://github.com/golom-protocol/contracts/commit/366c0455547041003c28f21b9afba48dc33dc5c7#diff-63895480b947c0761eff64ee21deb26847f597ebee3c024fb5aa3124ff78f6ccR154

[LSDan (judge) commented](#):

> Given how many upgrades I'm making on this, I figured a comment on my reasoning was in order. In many contests, this would be considered low risk. While unlikely to occur without warning, it is well-documented and so very well might occur at some point in the foreseeable future. With Golom's implementation, the entire functionality of the protocol would break if the gas price were to rise, resulting in a need to relaunch/redeploy. The extreme nature of this disruption offsets the other factors normally considered and is why I consider it to be a medium risk in this contest.

[M-02] Use `safeTransferFrom` Instead of `transferFrom` for ERC721

Submitted by TomJ, also found by __141345__, _Adam, 0x4non, 0x52, 0xDjango, 0xf15ers, 0xNazgul, 0xsanson, 8olidity, apostle0x01, arcoun, benbaessler, bin2chen, Bnke0x0, brgltd, cccz, CertoraInc, Ch_301, Chom, cloudjunky, cryptonue, djxploit, Dravee, ellahi, erictee, GalloDaSballo, GimelSec, hansfriese, i0001, JC, Jujic, Kenshin, Kumpa, Lambda, M0ndoHEHE, minhquanym, oyc_109, PaludoX0, peritoflores, rbserver, reassor, RedOneN, rokinot, rotcivegaf, Ruhum, saian, shenwilly, Sm4rty, sseefried, Treasure-Seeker, TrungOre, Twpony, and Waze

[GolomTrader.sol#L236](#)

Use of `transferFrom` method for ERC721 transfer is discouraged and recommended to use safeTransferFrom whenever possible by OpenZeppelin.
This is because `transferFrom()` cannot check whether the receiving address know how to handle ERC721 tokens.

In the function shown at below PoC, ERC721 token is sent to `msg.sender` with the `transferFrom` method.
If this `msg.sender` is a contract and is not aware of incoming ERC721 tokens, the sent token could be locked up in the contract forever.

Reference: https://docs.openzeppelin.com/contracts/3.x/api/token/erc721

Proof of Concept

```
    GolomTrader.sol:236:            ERC721(o.collection).transferFro
```

Recommended Mitigation Steps

I recommend to call the `safeTransferFrom()` method instead of `transferFrom()` for NFT transfers.

0xsaruman (Golom) confirmed, but disagreed with severity

0xsaruman (Golom) resolved and commented:

> Resolved https://github.com/golom-protocol/contracts/commit/366c0455547041003c28f21b9afba48dc33dc5c7#diff-63895480b947c0761eff64ee21deb26847f597ebee3c024fb5aa3124ff78f6ccR238

[M-03] Voter in `VoteEscrowCore` can permanently lock user tokens

Submitted by scaraven

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L873-L876

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L883-L886

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L894

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L538

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L1008

A malicious voter can arbitrarily increase the number of `attachments` or set the `voted` status of a token to true. This prevents the token from being withdrawn, merged or transfered thereby locking the tokens into the contract for as long as the voter would like.

I submitted this is as a medium severity because it has external circumstances (a malicious voter) however has a very high impact if it does occur.

Proof of Concept

1. A user creates a lock for their token and deposits it into the VoteEscrowDelegate/Core contract.
2. The malicious voter then calls either `voting()` or `attach()` thereby preventing the user withdrawing their token after the locked time bypasses

Tools Used

VS Code

Recommended Mitigation Steps

I have not seen any use of `voting()` or `attach()` in any of the other contracts so it may be sensible to remove those functions altogether. On the other hand, setting voter to be smart contract which is not malicious offsets this problem.

zeroexdead (Golom) confirmed

zeroexdead (Golom) commented:

> Removed Voter: https://github.com/golom-protocol/contracts/commit/03572010ef868597310f4736c91aacf3aa044ce9

0xsaruman (Golom) resolved

[M-04] `VoteEscrowCore.safeTransferFrom` does not check correct magic bytes returned from receiver contract's `onERC721Received` function

Submitted by sseefried, also found by 0x4non, arcoun, berndartmueller, cccz, csanuragjain, IllIllI, Jmaxmanblue, JohnSmith, Lambda, minhquanym, rbserver, and rotcivegaf

ERC721.sol#L395-L417

While `VoteEscrowCore.safeTransferFrom` does try to call `onERC721Received` on the receiver it does not check the for the required "magic bytes" which is `IERC721.onERC721received.selector` in this case. See OpenZeppelin docs for more information.

It's quite possible that a call to `onERC721Received` could succeed because the contract had a `fallback` function implemented, but the contract is not ERC721 compliant.

The impact is that NFT tokens may be sent to non-compliant contracts and lost.

Proof of Concept

Lines 604 - 605 are:

```
    try IERC721Receiver(_to).onERC721Received(msg.sender, _from, _to
        bytes memory reason
```

but they should be:

```
    try IERC721Receiver(to).onERC721Received(_msgSender(), from, tok
        return retval == IERC721Receiver.onERC721Received.selector;
    } catch (bytes memory reason)
```

Recommended Mitigation Steps

Implement `safeTransferReturn` so that it checks the required magic bytes: `IERC721Receiver.onERC721Received.selector`.

[zeroexdead (Golom) confirmed]()

[zeroexdead (Golom) commented]():

> Fixed.
> Ref: https://github.com/golom-protocol/contracts/commit/19ba6e83892e24b859f081525c7e0f751f5e7ebb

[0xsaruman (Golom) resolved, but disagreed with severity]()

[M-05] Replay attack in case of hard fork

Submitted by codexploder, also found by 0x1f8b, 0xNineDec, 0xsanson, berndartmueller, chatch, RustyRabbit, teddav, and Treasure-Seeker

[GolomTrader.sol#L98]()

If there is ever a hardfork for Golom then `EIP712_DOMAIN_TYPEHASH` value will become invalid. This is because the chainId parameter is computed in constructor. This means even after hard fork chainId would remain same which is incorrect and could cause possible replay attacks

Proof of Concept

1. Observe the constructor

```
constructor(address _governance) {
        // sets governance as owner
        _transferOwnership(_governance);

        uint256 chainId;
        assembly {
            chainId := chainid()
        }

        EIP712_DOMAIN_TYPEHASH = keccak256(
            abi.encode(
                keccak256('EIP712Domain(string name,string versi
                keccak256(bytes('GOLOM.IO')),
                keccak256(bytes('1')),
                chainId,
                address(this)
            )
        );
    }
```

2. As we can see the chainId is derived and then hardcoded in `EIP712_DOMAIN_TYPEHASH`
3. This means even after hard fork, `EIP712_DOMAIN_TYPEHASH` value will remain same and point to incorrect chainId

Recommended Mitigation Steps

The EIP712_DOMAIN_TYPEHASH variable should be recomputed everytime by placing current value of chainId.

0xsaruman (Golom) confirmed, but disagreed with severity

0xsaruman (Golom) resolved and commented:

> Resolved https://github.com/golom-protocol/contracts/commit/d8a24442b8f3a764139e312ed393e5d5ffb7e596

LSDan (judge) commented:

> I'm going to leave this as a medium risk. It would be a very high-impact scenario, but it relies on the external factor of a hard fork. That said, hard forks can and do happen.

[M-06] Orders with `tokenAmt` of `type(uint256).max` cannot be cancelled by `GolomTrader.sol#cancelOrder`

Submitted by 0x52, also found by GimelSec

GolomTrader.sol#L312-L317

Order unable to be cancelled by `cancelOrder`.

Proof of Concept

```
        filled[hashStruct] = o.tokenAmt + 1;
```

`cancelOrder` will overflow in the line shown above if `o.tokenAmt` is `type(uint256).max` causing the transaction to always revert for that order.

Recommended Mitigation Steps

I don't see any reason why 1 should be added to `o.tokenAmt`, change to:

```
        filled[hashStruct] = o.tokenAmt;
```

0xsaruman (Golom) acknowledged, but disagreed with severity

[M-07] veNFT withdraw and merge fail for approved callers

Submitted by horsefacts, also found by berndartmueller, csanuragjain, GalloDaSballo, hansfriese, IllIllI, kenzo, minhquanym, and rotcivegaf

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L893-L908

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L1004-L1030

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L1226-L1236

Golom is impacted by a known issue with the veNFT contract that causes the `merge` and `withdraw` functions to revert when called by an approved spender rather than the token owner.

`merge` and `withdraw` may both be called by either the token owner or an approved spender. Note that both of these functions check `_isApprovedOrOwner`:

`VoteEscrowCore#merge`

```
    function merge(uint256 _from, uint256 _to) external {
        require(attachments[_from] == 0 && !voted[_from], 'attac
        require(_from != _to);
        require(_isApprovedOrOwner(msg.sender, _from));
        require(_isApprovedOrOwner(msg.sender, _to));

        LockedBalance memory _locked0 = locked[_from];
        LockedBalance memory _locked1 = locked[_to];
        uint256 value0 = uint256(int256(_locked0.amount));
        uint256 end = _locked0.end >= _locked1.end ? _locked0.er

        locked[_from] = LockedBalance(0, 0);
        _checkpoint(_from, _locked0, LockedBalance(0, 0));
        _burn(_from);
        _deposit_for(_to, value0, end, _locked1, DepositType.MEI
    }
```

`VoteEscrowCore#withdraw`

```
    /// @notice Withdraw all tokens for `_tokenId`
    /// @dev Only possible if the lock has expired
    function withdraw(uint256 _tokenId) external nonreentrant {
        assert(_isApprovedOrOwner(msg.sender, _tokenId));
        require(attachments[_tokenId] == 0 && !voted[_tokenId],

        LockedBalance memory _locked = locked[_tokenId];
        require(block.timestamp >= _locked.end, "The lock didn't
        uint256 value = uint256(int256(_locked.amount));

        locked[_tokenId] = LockedBalance(0, 0);
        uint256 supply_before = supply;
        supply = supply_before - value;

        // old_locked can have either expired <= timestamp or ze
        // _locked has only 0 end
        // Both can have >= 0 amount
        _checkpoint(_tokenId, _locked, LockedBalance(0, 0));

        assert(IERC20(token).transfer(msg.sender, value));

        // Burn the NFT
        _burn(_tokenId);

        emit Withdraw(msg.sender, _tokenId, value, block.timesta
        emit Supply(supply_before, supply_before - value);
    }
```

However, both functions make internal calls to `_burn`, which does **not** handle the case of an approved caller correctly. The call to `_removeTokenFrom` on L1234 passes `msg.sender` rather than the token `owner`, which will revert:

`VoteEscrowCore#_burn`

```
    function _burn(uint256 _tokenId) internal {
        require(_isApprovedOrOwner(msg.sender, _tokenId), 'calle

        address owner = ownerOf(_tokenId);

        // Clear approval
        approve(address(0), _tokenId);
        // Remove token
```

```
            _removeTokenFrom(msg.sender, _tokenId);
            emit Transfer(owner, address(0), _tokenId);
        }
```

Impact

Approved callers cannot `merge` or `withdraw` veNFTs. `merge` and `withdraw` may only be called by the token owner.

Recommended Mitigation Steps

Update `_burn` to pass token owner address rather than `msg.sender`:

```
        function _burn(uint256 _tokenId) internal {
            require(_isApprovedOrOwner(msg.sender, _tokenId), 'calle

            address owner = ownerOf(_tokenId);

            // Clear approval
            approve(address(0), _tokenId);
            // Remove token
            _removeTokenFrom(owner, _tokenId);
            emit Transfer(owner, address(0), _tokenId);
        }
```

zeroexdead (Golom) confirmed

zeroexdead (Golom) commented:

> Removed `merge` and fixed withdraw here: https://github.com/golom-protocol/contracts/commit/c79913ec08ba2dca87a22f1bc6fe47f65f7b4202

0xsaruman (Golom) resolved

[M-08] Pre-check is not correct

Submitted by 0xSky

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L342

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L397

`fillCriteriaBid` can be reverted due to the pre-check while it can work.

Proof of Concept

When `refererrAmt > 0` and `referrer` address is not set (is 0), `(o.totalAmt - protocolfee - o.exchange.paymentAmt - o.prePayment.paymentAmt) * amount - p.paymentAmt >= 0` and `o.totalAmt < o.exchange.paymentAmt + o.prePayment.paymentAmt + o.refererrAmt` can hold true at the same time.

It is when `o.refererrAmt > (p.paymentAmt + protocolfee) / amount`.
In that case, `_settleBalances` can work, but fillCriteriaBid will be reverted due to the check in line 342.

Recommended Mitigation Steps

I think `require(o.totalAmt >= o.exchange.paymentAmt + o.prePayment.paymentAmt)` is correct.

0xsaruman (Golom) confirmed, but disagreed with severity and commented:

> Very small chance of both conditions happening, `o.refererrAmt > (p.paymentAmt + protocolfee) / amount` and referrer address is 0.

> Resolved https://github.com/golom-protocol/contracts/commit/c15fa96271d8cf764274271eee649c79ca1b1f7d

LSDan (judge) commented:

> Nice catch.

> This tracks as a medium for me... it breaks protocol functionality given external factors.

[M-09] `GolomToken.sol` doesn't contain a function to mint treasury tokens

Submitted by 0x52

[GolomToken.sol#L14-L73]()

Potential downtime in GolomTrader

Proof of Concept

GolomToken.sol doesn't have a function to mint the treasury tokens as specified in the docs (https://docs.golom.io/tokenomics-and-airdrop). In order for these tokens to be minted, the minter would have to be changed via `setMinter()` and `executeSetMinter()` to a contract that can mint the treasury tokens. Because of the 24 hour timelock, this would lead to downtime for GolomTrader.sol if trading has already begun. This is because GolomTrader.sol calls `RewardDistributor.sol#addFees` each time there is a filled order. When the epoch changes, RewardDistributor.sol will try to call the mint function in GolomToken.sol. Because of the timelock, there will be at least a 24 hours period where RewardDistributor.sol is not the minter and doesn't have the permission to mint. This means that during that period all trades will revert.

Recommended Mitigation Steps

Add a function to GolomToken.sol to mint the treasury tokens similar to the `mintAirdrop()` and `mintGenesisReward()` functions.

[0xsaruman (Golom) confirmed, but disagreed with severity]()

[0xsaruman (Golom) resolved and commented]():

> https://github.com/golom-protocol/contracts/commit/746507ea6f71a017be178f7eeb66d2dbf92a4524

[M-10] Delegated NFTs that are withdrawn while still delegated will remain delegated even after burn

Submitted by 0x52, also found by berndartmueller, IllIllI, kenzo, and rotcivegaf

[VoteEscrowCore.sol#L1226-L1236]

Burn NFTs remained delegated causing bloat and wasting gas.

Proof of Concept

VoteEscrowDelegation.sol doesn't change the withdraw or _burn functions inherited from VoteEscrowCore.sol. These functions are ignorant of the delegation system and don't properly remove the delegation when burning an NFT. The votes for the burned NFT will be removed but the reference will still be stored in the delegation list where it was last delegated. This creates a few issues. 1) It adds bloat to both getVotes and getPriorVotes because it adds a useless element that must be looped through. 2) The max number of users that can delegate to another NFT is 500 and the burned NFT takes up one of those spots reducing the number of real users that can delegate. 3) Adds gas cost when calling removeDelegation which adds gas cost to _transferFrom because removeElement has to cycle through a larger number of elements.

Recommended Mitigation Steps

Override _burn in VoteEscrowDelegation and add this.removeDelegation(_tokenId), similar to how it was done in _transferFrom.

[zeroexdead (Golom) confirmed]

[zeroexdead (Golom) commented]:

> Fixed.
> Ref: https://github.com/golom-protocol/contracts/commit/a30a50abe1aa677374bdbf68e1e81d80e1545563

[0xsaruman (Golom) resolved]

[LSDan (judge) commented]:

> I agree with the wardens and sponsor who rate this as medium. It does negatively impact the functioning of the protocol, but none of the reporting wardens have shown how it can be used as a direct attack vector IMO.

[M-11] When `MIN_VOTING_POWER_REQUIRED` is changed, previous votes are not affected

Submitted by cccz

When `MIN_VOTING_POWER_REQUIRED` is changed, tokenIDs with votes lower than `MIN_VOTING_POWER_REQUIRED` will not be able to vote through the delegate function, but previous votes will not be affected.
Since `MIN_VOTING_POWER_REQUIRED` is mainly used to reduce the influence of spam users, changing this value should affect previous votes.

Proof of Concept

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowDelegation.sol#L168-L194

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowDelegation.sol#L260-L262

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowDelegation.sol#L73-L74

Recommended Mitigation Steps

In the getPriorVotes and getVotes functions, when the balance corresponding to tokenId is less than MIN*VOTING*POWER_REQUIRED, the value of votes will not be increased

```
    function getVotes(uint256 tokenId) external view returns (ui
        uint256[] memory delegated = _getCurrentDelegated(tokenI
        uint256 votes = 0;
        for (uint256 index = 0; index < delegated.length; index+
+         if(this.balanceOfNFT(delegated[index]) >= MIN_VOTING_P
            votes = votes + this.balanceOfNFT(delegated[index]);
+       }
        }
        return votes;
    }


    /**
     * @notice Determine the prior number of votes for an accour
     * @dev Block number must be a finalized block or else this
     * @param tokenId The address of the account to check
     * @param blockNumber The block number to get the vote balar
     * @return The number of votes the account had as of the giv
     */
    function getPriorVotes(uint256 tokenId, uint256 blockNumber)
        require(blockNumber < block.number, 'VEDelegation: not y
        uint256[] memory delegatednft = _getPriorDelegated(toker
        uint256 votes = 0;
        for (uint256 index = 0; index < delegatednft.length; inc
+         if(this.balanceOfAtNFT(delegatednft[index], blockNumbe
            votes = votes + this.balanceOfAtNFT(delegatednft[inc
+         }
        }
        return votes;
    }
```

[zeroexdead (Golom) confirmed, but disagreed with severity]

[zeroexdead (Golom) commented]:

> When calling we `getVotes()` and `getPriorVotes()` we're considering `MIN_VOTING_POWER_REQUIRED`.
> Reference: https://github.com/golom-protocol/contracts/commit/db650729b0805ec19906a0ea11de6af7a53ac382

[0xsaruman (Golom) resolved]

[LSDan (judge) decreased severity to Medium and commented]:

> Downgrading this to medium. Assets are not at direct risk.

[M-12] Some setters' timelock can be bypassed

Submitted by zzzitron, also found by berndartmueller, GimelSec, GiveMeTestEther, and sseefried

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/governance/GolomToken.sol#L58-L72

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/core/GolomTrader.sol#L444-L457

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L298-L311

MED - Function could be impacted

As the timelock does not work as supposed to work, the owner of the contract can bypass timelock.

- Affected Functions:
  - `GolomToken`: `setMinter`, `executeSetMinter`
  - `GolomTrader`: `setDistributor`, `executeSetDistributor`
  - `RewardDistributor`: `addVoteEscrow`, `executeAddVoteEscrow`

Proof of Concept

- GolomTrader::it can bypass timelock poc
- GolomToken::setMinter it should set the minter with timelock poc

The first poc shows to bypass timelock for `GolomTrader::setDistributor`. The same logic applies for the `RewardDistributor::addVoteEscrow`.
0. The `setDistributor` was called once in the beforeEach block to set the initial distributor. For this exploit to work, the `setDistributor` should be called only

once. If `setDistributor` was called more than once, one can set the distributor to zero address (with timelock like in the `GolomToken` case, then set to a new distributor after that)

1. reset distributor to zero address without timelock by calling `executeSetDistributor`
2. set a new distributor without timelock by calling `setDistributor`
3. Rinse and repeat: as long as `setDistributor` is not called multiple times in row, the owner can keep setting distributor without timelock.

A little bit different variation of timelock bypass was found in the `GolomToken`. Although the owner should wait for the timelock to set the minter to zero address, but after that, the owner can set to the new minter without waiting for a timelock. Since the meaning of timelock is to let people know the new minter's implementation, if the owner can bypass that, the timelock is almost meaningless. The exploitation steps: [the second proof of concept]()

1. call `setMineter` with zero address
2. wait for the timelock
3. call `executeSetMineter` to set the minter to zero address
4. now the onwer can call `setMineter` with any address and call `executeSetMinter` without waiting for the timelock

The owner can call `executeSetdistributor` even though there is no `pendingDistributor` set before. Also, `setDistributor` sets the new distributor without timelock when the existing distributor's address is zero.

```
// GolomTrader
// almost identical logic was used in `RewardDistributor` to ad
// similar logic was used in `GolomToken` to `setMineter`
```

```
    function setDistributor(address _distributor) external
        if (address(distributor) == address(0)) {
            distributor = Distributor(_distributor);
        } else {
            pendingDistributor = _distributor;
```

```
                distributorEnableDate = block.timestamp + 1 days
            }
        }

        /// @notice Executes the set distributor function after
        function executeSetDistributor() external onlyOwner {
            require(distributorEnableDate <= block.timestamp, 'r
            distributor = Distributor(pendingDistributor);
        }
```

Recommended Mitigation Steps

To mitigate, execute functions can check whether pendingDistributor is not zero. It will ensure that the setters are called before executing them, as well as prevent to set to zero addresses.

0xsaruman (Golom) disputed and commented:

> call setMineter with zero address
> wait for the timelock

> This alone will trigger awareness that something malicious is happening and since timelock is there people have time to get out.

0xsaruman (Golom) confirmed and commented:

> The second POC is valid.

0xsaruman (Golom) resolved and commented:

> Removed all secondary time locks in the contract and only using the primary timelock that will be behind the owner.

> https://github.com/golom-protocol/contracts/commit/366c0455547041003c28f21b9afba48dc33dc5c7#diff-94d75c3059a714c355bd15d139c30d4f9899df283d29717622ffd5c930445499R59

[M-13] Rewards owed burned NFT in `RewardDistributor.sol` become irretrievable

Submitted by 0x52, also found by kyteg

[RewardDistributor.sol#L172-L210]()

Rewards owed burned NFT are permanently locked.

Proof of Concept

```
    function _burn(uint256 _tokenId) internal {
        require(_isApprovedOrOwner(msg.sender, _tokenId), 'caller is

        address owner = ownerOf(_tokenId);

        // Clear approval
        approve(address(0), _tokenId);
        // Remove token
        _removeTokenFrom(msg.sender, _tokenId);
        emit Transfer(owner, address(0), _tokenId);
    }

    function _removeTokenFrom(address _from, uint256 _tokenId) inter
        // Throws if `_from` is not the current owner
        assert(idToOwner[_tokenId] == _from);
        // Change the owner
        idToOwner[_tokenId] = address(0);
        // Update owner token index tracking
        _removeTokenFromOwnerList(_from, _tokenId);
        // Change count tracking
        ownerToNFTokenCount[_from] -= 1;
    }
```

After an NFT is burned, owner of token is set to `address(0)`.

```
        rewardToken.transfer(tokenowner, reward);
```

This causes issues in multiStakerClaim L208. GOLOM uses OZ's implementation of ERC20 which doesn't allow tokens to be sent to `address(0)`. Because the "owner" of the burned NFT is `address(0)` multiStakerClaim will always revert when called for a burned NFT trapping rewards in contract forever.

Recommended Mitigation Steps

Implement a clawback clause inside the multiStakerClaim function. If the token is burned (i.e. owned by address(0)) the rewards should be transferred to different address. These rewards could be claimed to the treasury or burned, etc.

```
        if (tokenowner == address(0){
            rewardToken.transfer(treasury, reward);
            weth.transfer(treasury, rewardEth);
        }
```

[0xsaruman (Golom) confirmed, but disagreed with severity and commented]:

> I think its QA because burning NFT means the owner doesn't want anything to do with rewards or the NFT anymore.

[M-14] `VoteEscrowDelegation._transferFrom` can only be executed by the token owner

Submitted by GimelSec, also found by GalloDaSballo, kebabsec, and kenzo

`VoteEscrowDelegation._transferFrom` should be successfully executed if `msg.sender` is the current owner, an authorized operator, or the approved address. `removeDelegation` is called in `_transferFrom`. `removeDelegation` only accepts the token owner. Thus, `_transferFrom` can only be executed by the token owner.

Proof of Concept

`removeDelegation` is called in `_transferFrom`

https://github.com/code-423n4/2022-07-golom/blob/main/contracts/vote-escrow/VoteEscrowDelegation.sol#L242

```
function _transferFrom(
    address _from,
    address _to,
    uint256 _tokenId,
    address _sender
) internal override {
    require(attachments[_tokenId] == 0 && !voted[_tokenId],

    // remove the delegation
    this.removeDelegation(_tokenId);

    // Check requirements
    require(_isApprovedOrOwner(_sender, _tokenId));
    …
}
```

However, `removeDelegation` only accept the token owner https://github.com/code-423n4/2022-07-golom/blob/main/contracts/vote-escrow/VoteEscrowDelegation.sol#L211

```
function removeDelegation(uint256 tokenId) external {
    require(ownerOf(tokenId) == msg.sender, 'VEDelegation: N
    uint256 nCheckpoints = numCheckpoints[tokenId];
    Checkpoint storage checkpoint = checkpoints[tokenId][nCh
    removeElement(checkpoint.delegatedTokenIds, tokenId);
    _writeCheckpoint(tokenId, nCheckpoints, checkpoint.deleg
}
```

Recommended Mitigation Steps

Fix the permission control in `removeDelegation`.

[zeroexdead (Golom) confirmed]()

[zeroexdead (Golom) commented]():

> Changed the `external` function to `public`. Users address will be passed as `msg.sender` now.

> https://github.com/golom-protocol/contracts/commit/10ec920765a5ee2afc2fe269d32ea9138d1156b6

0xsaruman (Golom) resolved

[M-15] Griefer can minimize delegatee's voting power

Submitted by 0xDjango, also found by 0x52, 0xsanson, kenzo, MEP, and simon135

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowDelegation.sol#L99

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowDelegation.sol#L71-L89

Similar to a previous submission, there are no checks preventing against delegating the same lock NFT multiple times. This opens an avenue to an expensive but potentially profitable griefing attack where the malicious user fills the victim's delegated token array with minimum voting power. The attacker can ensure that a delegatee has 0 voting power.

Proof of Concept

Taking a look at the `delegate()` function below, there are no checks that a lock NFT has not already been delegated. Therefore, an attacker can delegate their token with minimum voting power (threshold initialized with value 0) to the victim.

```
    function delegate(uint256 tokenId, uint256 toTokenId) extern
        require(ownerOf(tokenId) == msg.sender, 'VEDelegation: N
        require(this.balanceOfNFT(tokenId) >= MIN_VOTING_POWER_F


        delegates[tokenId] = toTokenId;
        uint256 nCheckpoints = numCheckpoints[toTokenId];


        if (nCheckpoints > 0) {
            Checkpoint storage checkpoint = checkpoints[toTokenI
```

```
                checkpoint.delegatedTokenIds.push(tokenId);
                _writeCheckpoint(toTokenId, nCheckpoints, checkpoint
            } else {
                uint256[] memory array = new uint256[](1);
                array[0] = tokenId;
                _writeCheckpoint(toTokenId, nCheckpoints, array);
            }


            emit DelegateChanged(tokenId, toTokenId, msg.sender);
        }
```

There is a limit of 500 delegated tokens per delegatee. Therefore, the attacker can ensure minimum voting power if they delegate a worthless token 500 times to the victim:

```
        function _writeCheckpoint(
            uint256 toTokenId,
            uint256 nCheckpoints,
            uint256[] memory _delegatedTokenIds
        ) internal {
            require(_delegatedTokenIds.length < 500, 'VVDelegation:
```

A more likely scenario would be as follows:

- A proposal is live.
- Users delegate their voting power to addresses of their choosing.
- A and B are around the same voting power.
- A and B both have 400 delegatees.
- Malicious address A delegates minimum voting power 100 times to fill B’s array to 500.
- Address A can self-delegate just a bit more to obtain more voting power.

Recommended Mitigation Steps

Firstly, removing the ability to delegate the same lock NFT would make this griefing attack much more expensive. Even if that is patched, a griefing attack is still possible by simply creating more locks and delegating them all once.

I believe that removing the 500 delegated token limit would prove to mitigate this issue.

zeroexdead (Golom) confirmed

zeroexdead (Golom) commented:

> We plan to keep sufficiently high `MIN_VOTING_POWER_REQUIRED` to prevent spam.

zeroexdead (Golom) commented:

> Removed the ability to delegate same NFT. If user is trying to delegate same NFT, the old delegation will be removed.

> Reference: https://github.com/golom-protocol/contracts/commit/c74d95b4105eeb878d2781982178db5ca08a1a9b

LSDan (judge) commented:

> I agree with the ranking of medium. This is a direct attack vector, but it's unlikely to be used.

[M-16] `GolomTrader`: `validateOrder` function does not check if ecrecover return value is 0

Submitted by cccz, also found by 0x1f8b, 0xHarry, AuditsAreUS, djxploit, jayjonah8, joestakey, and teddav

The validateOrder function of GolomTrader calls the Solidity ecrecover function directly to verify the given signatures. The return value of ecrecover may be 0, which means the signature is invalid, but the check can be bypassed when signer is 0.

Proof of Concept

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L176-L177

Recommended Mitigation Steps

Use the recover function from OpenZeppelin's ECDSA library for signature verification.

kenzo (warden) commented:

> Seems invalid or QA at best. No impact on protocol as far as I see, invalid orders from "address 0" will revert.
> In fillAsk if the `o.signer` is address 0, the function will try to pull tokens from address 0 and will fail.
> In fillBid/criteria, function will try to transfer msg.sender's tokens to address 0 and pull weth from address 0. So will fail.

0xsaruman (Golom) disputed

LSDan (judge) commented:

> This is valid as a medium risk. It opens a griefing attack where a bad actor spams any system that relies on this function. The fact that the fill will fail while the order appears valid is specifically what makes this griefing attack possible.

[M-17] NFTs that don't have a checkpoint can't be transferred

Submitted by 0x52, also found by 0xsanson

Submitting as high risk because it breaks a fundamental operation (transferring) for a large number of tokens.

Proof of Concept

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowDelegation.sol#L212-L213

```
        uint256 nCheckpoints = numCheckpoints[tokenId];
        Checkpoint storage checkpoint = checkpoints[tokenId][nCheckp
```

L242 of `transferFrom()` calls `removeDelegation()` with the tokenId of the token being transferred. For tokens that don't have any checkpoints, L212 will return 0. This was cause an underflow error and revert in L213.

Recommended Mitigation Steps

Make removeDelegation simply return if `nCheckpoints = 0.`

[kenzo (warden) commented]:

> Unsure if high risk, but warden correctly identified the issue (that some others didn't) that the underflow in `removeDelegation` will prevent tokens from being transferred.

[zeroexdead (Golom) disagreed with severity]

[LSDan (judge) decreased severity to Medium and commented]:

> Marking this as a medium risk because it only temporarily breaks functionality. The workaround would be to delegate the token and then transfer it, making the impact aggravating but ultimately minimal.

[M-18] `fillAsk()` Allows for `msg.value` to be larger than require locking the excess in the contract

Submitted by AuditsAreUS, also found by 0xSky, arcoun, bin2chen, cccz, CertoraInc, codexploder, cryptonue, dipp, GimelSec, GiveMeTestEther, Green, horsefacts, jayphbee, joestakey, Lambda, minhquanym, obront, peritoflores, rbserver, reassor, rotcivegaf, Ruhum, RustyRabbit, scaraven, Treasure-Seeker, Twpony, and ych18

[GolomTrader.sol#L217]

It is possible to send a higher `msg.value` than is required to `fillAsk()`. The excess value that is sent will be permanently locked in the contract.

Proof of Concept

There is only one check over `msg.value` and it is that it’s greater than `o.totalAmt * amount + p.paymentAmt` . As seen in the following code snippet from #217.

```
            require(msg.value >= o.totalAmt * amount + p.paymentAmt,
```

The issue here is that the contract will only ever spend exactly `o.totalAmt * amount + p.paymentAmt` . Hence if `msg.value` is greater than this then the excess value will be permanently locked in the contract.

Recommended Mitigation Steps

To avoid this issue consider enforcing a strict equality.

```
            require(msg.value == o.totalAmt * amount + p.paymentAmt,
```

0xsaruman (Golom) confirmed

0xsaruman (Golom) disagreed with severity and commented:

> Resolved: https://github.com/golom-protocol/contracts/commit/366c0455547041003c28f21b9afba48dc33dc5c7#diff-63895480b947c0761eff64ee21deb26847f597ebee3c024fb5aa3124ff78f6ccR217

> Disagree with severity cause it’s user choice to send more.

LSDan (judge) commented:

> I agree with this being a medium. It opens up the potential for griefing attacks and all sorts of other issues that may be beyond the scope of “the user decided to send excess funds”. Further, it’s common for contracts to return excess funds, so the user may reasonably expect this behaviour.

Low Risk and Non-Critical Issues

For this contest, 130 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by IllIllI received the top score from the judge.

The following wardens also submitted reports: __141345__, _Adam, 0x1f8b, 0x4non, 0x52, 0xA5DF, 0xackermann, 0xc0ffEE, 0xDjango, 0xf15ers, reassor, RedOneN, robee, Rohan16, rokinot, Rolezn, rotcivegaf, Ruhum, RustyRabbit, sach1r0, 0xLovesleep, saian, saneryee, sashik_eth, scaraven, shenwilly, simon135, Sm4rty, Soosh, sseefried, zzzitron, 0xmatt, StErMi, StyxRave, supernova, Tadashi, teddav, TomJ, Treasure-Seeker, TrungOre, Waze, ych18, 0xNazgul, zuhaibmohd, 0xNineDec, 0xsanson, 0xSmartContract, 0xsolstars, 8olidity, ajtra, ak1, apostle0x01, arcoun, asutorufos, async, AuditsAreUS, Bahurum, benbaessler, berndartmueller, bin2chen, Bnke0x0, brgltd, c3phas, carlitox477, CertoraInc, Ch_301, chatch, Chom, codetilda, codexploder, cRat1st0s, CRYP70, CryptoMartian, cryptonue, cryptphi, csanuragjain, cthulhu_cult, Deivitto, delfin454000, DevABDee, dipp, dirk_y, djxploit, Dravee, ElKu, ellahi, exd0tpy, fatherOfBlocks, Franfran, Funen, GalloDaSballo, GimelSec, giovannidisiena, GiveMeTestEther, Green, hansfriese, horsefacts, hyh, idkwhatimdoing, indijanc, jayfromthe13th, jayphbee, JC, Jmaxmanblue, joestakey, JohnSmith, Jujic, Junnon, Kenshin, kenzo, Krow10, Kumpa, kyteg, Lambda, lucacez, luckypanda, Maxime, MEP, mics, MiloTruck, minhquanym, Mohandes, NoamYakov, obront, oyc_109, pedr02b2, Picodes, rajatbeladiya, *and* rbserver.

Low Risk Issues

	Issue	Instances
[L-01]	Only a billion checkpoints available	1
[L-02]	Don't use `payable.transfer()` / `payable.send()`	1
[L-03]	Unused/empty `receive()` / `fallback()` function	4
[L-04]	`require()` should be used instead of `assert()`	13
[L-05]	Self-delegation is not automatic	1

	Issue	Instances
[L-06]	Function may run out of gas	1
[L-07]	Vulnerable to cross-chain replay attacks due to static `DOMAIN_SEPARATOR` / `domainSeparator`	1
[L-08]	Wrong comment	1
[L-09]	Missing checks for `address(0x0)` when assigning values to `address` state variables	7
[L-10]	Missing event and or timelock for critical parameter change	3
[L-11]	Inconsistent spacing in comments	2
[L-12]	Typos	24
[L-13]	NatSpec is incomplete	19

Total: 78 instances over 13 issues

Non-critical Issues

	Issue	Instances
[N-01]	Consider addings checks for signature malleability	1
[N-02]	`ecrecover()` signature validity not checked	1
[N-03]	Boilerplate not replaced	2
[N-04]	Remove commented out code	1
[N-05]	Invalid/extraneous/optional function definitions in interface	4
[N-06]	Remove `include` for hardhat's console	1
[N-07]	Contract implements interface without extending the interface	1

	Issue	Instances
[N-08]	`require()` / `revert()` statements should have descriptive reason strings	31
[N-09]	`public` functions not called by the contract should be declared `external` instead	13
[N-10]	Non-assembly method available	2
[N-11]	`constant` s should be defined rather than using magic numbers	49
[N-12]	Numeric values having to do with time should use time units for readability	5
[N-13]	Large multiples of ten should use scientific notation (e.g. `1e6`) rather than decimal literals (e.g. `1000000`), for readability	2
[N-14]	Use a more recent version of solidity	2
[N-15]	Use scientific notation (e.g. `1e18`) rather than exponentiation (e.g. `10**18`)	2
[N-16]	Lines are too long	8
[N-17]	Inconsistent method of specifying a floating pragma	1
[N-18]	Variable names that consist of all capital letters should be reserved for `constant` / `immutable` variables	2
[N-19]	Non-library/interface files should use fixed compiler versions, not floating ones	1
[N-20]	File does not contain an SPDX Identifier	1
[N-21]	Event is missing `indexed` fields	7
[N-22]	Duplicated `require()` / `revert()` checks should be refactored to a modifier or function	14

Total: 151 instances over 22 issues

[L-1] Only a billion checkpoints available

A user can only have a billion checkpoints which, if the user is a DAO, may cause issues down the line, especially if the last checkpoint involved delegating and can thereafter not be undone

There is 1 instance of this issue:

```
File: contracts/contracts/VotingEscrow.sol

535:     mapping(uint => Point[1000000000]) public user_point_hi
```

https://github.com/code-423n4/2022-05-velodrome/blob/7fda97c570b758bbfa7dd6724a336c43d4041740/contracts/contracts/VotingEscrow.sol#L535

[L-02] Don’t use `payable.transfer()` / `payable.send()`

The use of `payable.transfer()` is **heavily frowned upon** because it can lead to the locking of funds. The `transfer()` call requires that the recipient is either an EOA account, or is a contract that has a `payable` callback. For the contract case, the `transfer()` call only provides 2300 gas for the contract to complete its operations. This means the following cases can cause the transfer to fail:

- The contract does not have a `payable` callback
- The contract’s `payable` callback spends more than 2300 gas (which is only enough to emit something)
- The contract is called through a proxy which itself uses up the 2300 gas Use OpenZeppelin’s `Address.sendValue()` instead

There is 1 instance of this issue:

```
File: contracts/core/GolomTrader.sol

154:                 payable(payAddress).transfer(payAmt); // royal
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/Gol

omTrader.sol#L154

[L-03] Unused/empty `receive()` / `fallback()` function

If the intention is for the Ether to be used, the function should call another function, otherwise it should revert (e.g. `require(msg.sender == address(weth))`)

There are 4 instances of this issue:

```
File: contracts/core/GolomTrader.sol

459:        fallback() external payable {}

461:        receive() external payable {}
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L459

```
File: contracts/rewards/RewardDistributor.sol

313:        fallback() external payable {}

315:        receive() external payable {}
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L313

[L-04] `require()` should be used instead of `assert()`

Prior to solidity version 0.8.0, hitting an assert consumes the **remainder of the transaction's available gas** rather than returning it, as `require()` / `revert()` do. `assert()` should be avoided even past solidity version 0.8.0 as its documentation states that "The assert function creates an error of type Panic(uint256). ... Properly functioning code should never create a Panic, not even on invalid external input. If this happens, then there is a bug in your contract which you should fix".

There are 13 instances of this issue:

```
File: contracts/vote-escrow/VoteEscrowCore.sol

493:            assert(idToOwner[_tokenId] == address(0));

506:            assert(idToOwner[_tokenId] == _from);

519:            assert(idToOwner[_tokenId] == _owner);

666:            assert(_operator != msg.sender);

679:            assert(_to != address(0));

861:                assert(IERC20(token).transferFrom(from, addres

977:            assert(_isApprovedOrOwner(msg.sender, _tokenId));

981:            assert(_value > 0); // dev: need non-zero value

991:            assert(_isApprovedOrOwner(msg.sender, _tokenId));

1007:           assert(_isApprovedOrOwner(msg.sender, _tokenId));

1023:           assert(IERC20(token).transfer(msg.sender, value));

1110:           assert(_block <= block.number);

1206:           assert(_block <= block.number);
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L493

[L-05] Self-delegation is not automatic

Unlike some of the other functions, `_mint()` isn't overridden to call `delegate()`, which means the user may forget to do so and will miss out

There is 1 instance of this issue:

```
File: /contracts/vote-escrow/VoteEscrowCore.sol

677      function _mint(address _to, uint256 _tokenId) internal
678          // Throws if `_to` is zero address
679          assert(_to != address(0));
680          // Add NFT. Throws if `_tokenId` is owned by someor
681          _addTokenTo(_to, _tokenId);
682          emit Transfer(address(0), _to, _tokenId);
683          return true;
684:     }
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L677-L684

[L-06] Function may run out of gas

Once the number of epochs grow to a large number, the array allocated will be large, and the number of iterations calling external functions on `ve` will also be large, leading to the function running out of gas

There is 1 instance of this issue:

```
File: /contracts/rewards/RewardDistributor.sol

215      function stakerRewards(uint256 tokenid) public view ret
216              uint256,
217              uint256,
218              uint256[] memory
219          ){
220          require(address(ve) != address(0), ' VE not added y
221
222          uint256 reward = 0;
223          uint256 rewardEth = 0;
224          uint256[] memory unclaimedepochs = new uint256[](ep
225          // for each epoch
226:         for (uint256 index = 0; index < epoch; index++) {
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards

/RewardDistributor.sol#L215-L224

[L-07] Vulnerable to cross-chain replay attacks due to static `DOMAIN_SEPARATOR` / `domainSeparator`

See this issue from a prior contest for details

There is 1 instance of this issue:

```
File: /contracts/core/GolomTrader.sol

101          EIP712_DOMAIN_TYPEHASH = keccak256(
102              abi.encode(
103                  keccak256('EIP712Domain(string name,string
104                  keccak256(bytes('GOLOM.IO')),
105                  keccak256(bytes('1')),
106                  chainId,
107                  address(this)
108              )
109:         );
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L101-L109

[L-08] Wrong comment

The function description and return values are incorrectly copied from another function

There is 1 instance of this issue:

```
File: /contracts/rewards/RewardDistributor.sol

252      /// @dev returns unclaimed rewards of an NFT, returns
253      /// @param addr the nft id to claim rewards for all ids
254      function traderRewards(address addr) public view return
255              uint256
256:             ){
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L252-L256

[L-09] Missing checks for `address(0x0)` when assigning values to `address` state variables

There are 7 instances of this issue:

```
File: contracts/core/GolomTrader.sol

448:                pendingDistributor = _distributor;
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L448

```
File: contracts/governance/GolomToken.sol

59:             pendingMinter = _minter;
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/governance/GolomToken.sol#L59

```
File: contracts/rewards/RewardDistributor.sol

81:             trader = _trader;

287:            pendingTrader = _trader;

303:                pendingVoteEscrow = _voteEscrow;
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L81

```
File: contracts/vote-escrow/VoteEscrowCore.sol

870:        voter = _voter;
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L870

```
File: contracts/vote-escrow/VoteEscrowDelegation.sol

53:         token = _token;
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowDelegation.sol#L53

[L-10] Missing event and or timelock for critical parameter change

Events help non-contract tools to track changes, and events prevent users from being surprised by changes

There are 3 instances of this issue:

```
File: contracts/core/GolomTrader.sol

444     function setDistributor(address _distributor) externa
445         if (address(distributor) == address(0)) {
446             distributor = Distributor(_distributor);
447         } else {
448             pendingDistributor = _distributor;
449             distributorEnableDate = block.timestamp + 1 da
450         }
451:    }
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/Gol

omTrader.sol#L444-L451

```
File: contracts/governance/GolomToken.sol

58        function setMinter(address _minter) external onlyOwne
59            pendingMinter = _minter;
60            minterEnableDate = block.timestamp + 1 days;
61:       }
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/governance/GolomToken.sol#L58-L61

```
File: contracts/vote-escrow/VoteEscrowCore.sol

868       function setVoter(address _voter) external {
869           require(msg.sender == voter);
870           voter = _voter;
871:      }
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L868-L871

[L-11] Inconsistent spacing in comments

Some lines use `// x` and some use `//x`. The instances below point out the usages that don't follow the majority, within each file

There are 2 instances of this issue:

```
File: contracts/core/GolomTrader.sol

181:          //deadline
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/Gol

omTrader.sol#L181

```
File: contracts/rewards/RewardDistributor.sol

99:             //console.log(block.timestamp,epoch,fee);
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L99

[L-12] Typos

There are 24 instances of this issue:

```
File: contracts/core/GolomTrader.sol

/// @audit succesful
53:             Payment exchange; // payment agreed by maker of th

/// @audit facilating
54:             Payment prePayment; // another payment , can be us

/// @audit usefull
60:             uint256 nonce; // nonce of order usefull for cance

/// @audit succesful
201:        /// @param p any extra payment that the taker of this

/// @audit succesful
278:        /// @param p any extra payment that the taker of this

/// @audit succesful
333:        /// @param p any extra payment that the taker of this

/// @audit succesfully
370:        /// @dev function to settle balances when a bid is fil

/// @audit succesful
374:        /// @param p any extra payment that the taker of this
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L53

```
File: contracts/rewards/RewardDistributor.sol

/// @audit epoc
61:        mapping(uint256 => uint256) public rewardTrader; // re

/// @audit epoc
/// @audit exhange
62:        mapping(uint256 => uint256) public rewardExchange; //

/// @audit epoc
63:        mapping(uint256 => uint256) public rewardLP; // rewarc

/// @audit epoc
64:        mapping(uint256 => uint256) public rewardStaker; // re

/// @audit upto
66:        mapping(uint256 => uint256) public claimedUpto; // epc

/// @audit upto
67:        mapping(uint256 => mapping(uint256 => uint256)) public

/// @audit facilated
95:        /// @dev Add fees contributed by the Seller of nft anc

/// @audit atleast
107:                // this assumes atleast 1 trade is done daily?

/// @audit begiining
/// @audit begining
111:                // emissions is decided by epoch begiining loc

/// @audit facilated
154:        // allows exchange that facilated the nft trades to cl
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L61

```
File: contracts/vote-escrow/VoteEscrowCore.sol

/// @audit blocktimes
267:    * and per block could be fairly bad b/c Ethereum changes

/// @audit Exeute
526:      /// @dev Exeute transfer of a NFT.

/// @audit Pevious
688:      /// @param old_locked Pevious locked amount / end lock
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L267

```
File: contracts/vote-escrow/VoteEscrowDelegation.sol

/// @audit Exeute
227:      /// @dev Exeute transfer of a NFT.
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowDelegation.sol#L227

[L-13] NatSpec is incomplete

There are 19 instances of this issue:

```
File: contracts/core/GolomTrader.sol

/// @audit Missing: '@return'
162       ///      OrderStatus = 3 , valid order
163       /// @param o the Order struct to be validated
164       function validateOrder(Order calldata o)
165           public
166           view
167           returns (
168               uint256,
169               bytes32,
170:                uint256

/// @audit Missing: '@param tokenId'
/// @audit Missing: '@param proof'
328         /// @dev function to fill a signed order of ordertype
329         ///      to send ether to that address on filling the
330         /// @param o the Order struct to be filled must be orc
331         /// @param amount the amount of times the order is to
332         /// @param referrer referrer of the order
333         /// @param p any extra payment that the taker of this
334         function fillCriteriaBid(
335             Order calldata o,
336             uint256 amount,
337             uint256 tokenId,
338             bytes32[] calldata proof,
339             address referrer,
340             Payment calldata p
341:        ) public nonReentrant {
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L162-L170

```
File: contracts/rewards/RewardDistributor.sol

/// @audit Missing: '@return'
213         /// @dev returns unclaimed rewards of an NFT, returns
214         /// @param tokenid the nft id to claim rewards for all
215         function stakerRewards(uint256 tokenid) public view re
216                 uint256,
217                 uint256,
218:                uint256[] memory

/// @audit Missing: '@return'
252         /// @dev returns unclaimed rewards of an NFT, returns
253         /// @param addr the nft id to claim rewards for all ic
254         function traderRewards(address addr) public view retur
255:                uint256

/// @audit Missing: '@return'
267         /// @dev returns unclaimed golom rewards of a trader
268         /// @param addr the nft id to claim rewards for all ic
269         function exchangeRewards(address addr) public view ret
```

```
270:                uint256
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L213-L218

```
File: contracts/vote-escrow/VoteEscrowCore.sol

/// @audit Missing: '@return'
366        /// @dev Interface identification is specified in ERC-
367        /// @param _interfaceID Id of the interface
368:       function supportsInterface(bytes4 _interfaceID) extern

/// @audit Missing: '@return'
396        ///      Throws if `_owner` is the zero address. NFTs
397        /// @param _owner Address for whom to query the balanc
398:       function _balance(address _owner) internal view returr

/// @audit Missing: '@return'
403        ///      Throws if `_owner` is the zero address. NFTs
404        /// @param _owner Address for whom to query the balanc
405:       function balanceOf(address _owner) external view retur

/// @audit Missing: '@return'
409        /// @dev Returns the address of the owner of the NFT.
410        /// @param _tokenId The identifier for an NFT.
411:       function ownerOf(uint256 _tokenId) public view returns

/// @audit Missing: '@return'
415        /// @dev Get the approved address for a single NFT.
416        /// @param _tokenId ID of the NFT to query the approva
417:       function getApproved(uint256 _tokenId) external view r

/// @audit Missing: '@return'
422        /// @param _owner The address that owns the NFTs.
423        /// @param _operator The address that acts on behalf c
424:       function isApprovedForAll(address _owner, address _ope

/// @audit Missing: '@return'
935        /// @param _lock_duration Number of seconds to lock tc
936        /// @param _to Address to deposit
937        function _create_lock(
938            uint256 _value,
            uint256 _lock_duration,
            address _to
        ) internal returns (uint256) {

/// @audit Missing: '@return'
        /// @param _lock_duration Number of seconds to lock to
        /// @param _to Address to deposit
        function create_lock_for(
            uint256 _value,
            uint256 _lock_duration,
            address _to
        ) external nonreentrant returns (uint256) {

/// @audit Missing: '@return'
        /// @param _value Amount to deposit
        /// @param _lock_duration Number of seconds to lock to
        function create_lock(uint256 _value, uint256 _lock_dur

/// @audit Missing: '@param _tokenId'
        /// @notice Deposit `_value` additional tokens for `_t
        /// @param _value Amount of tokens to deposit and add
        function increase_amount(uint256 _tokenId, uint256 _va

/// @audit Missing: '@param _tokenId'
        /// @notice Extend the unlock time for `_tokenId`
        /// @param _lock_duration New number of seconds until
        function increase_unlock_time(uint256 _tokenId, uint25

/// @audit Missing: '@return'
        /// @dev Returns current token URI metadata
        /// @param _tokenId Token ID to fetch URI for.
        function tokenURI(uint256 _tokenId) external view retu

/// @audit Missing: '@param t'
        /// @notice Calculate total voting power
        /// @dev Adheres to the ERC20 `totalSupply` interface
        /// @return Total voting power
        function totalSupplyAtT(uint256 t) public view returns
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L366-L368

[N-01] Consider addings checks for signature malleability

Use OpenZeppelin's `ECDSA` contract rather than calling `ecrecover()` directly

There is 1 instance of this issue:

```
File: /contracts/core/GolomTrader.sol

176         address signaturesigner = ecrecover(hash, o.v, o.r,
177         require(signaturesigner == o.signer, 'invalid signa
178         if (signaturesigner != o.signer) {
179             return (0, hashStruct, 0);
180:        }
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L176-L180

[N-02] `ecrecover()` signature validity not checked

`ecrecover()` returns the zero address if the signature is invalid. If the signer provided is also zero, then all incorrect signatures will be allowed

There is 1 instance of this issue:

```
File: /contracts/core/GolomTrader.sol

176         address signaturesigner = ecrecover(hash, o.v, o.r,
177         require(signaturesigner == o.signer, 'invalid signa
178         if (signaturesigner != o.signer) {
179             return (0, hashStruct, 0);
180:        }
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L176-L180

[N-03] Boilerplate not replaced

There are 2 instances of this issue:

```
File: /contracts/governance/GolomToken.sol

5:   /// @notice Explain to an end user what this does
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/governance/GolomToken.sol#L5

```
File: /contracts/vote-escrow/VoteEscrowDelegation.sol

68:      /// @notice Explain to an end user what this does
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowDelegation.sol#L68

[N-04] Remove commented out code

There is 1 instance of this issue:

```
File: /contracts/vote-escrow/VoteEscrowDelegation.sol

218      // /// @notice Remove delegation by user
219      // function removeDelegationByOwner(uint256 delegatedTc
220      //     require(ownerOf(ownerTokenId) == msg.sender, 'VE
221      //     uint256 nCheckpoints = numCheckpoints[delegatedT
222      //     Checkpoint storage checkpoint = checkpoints[dele
223      //     removeElement(checkpoint.delegatedTokenIds, dele
224      //     _writeCheckpoint(ownerTokenId, nCheckpoints, che
225:     // }
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowDelegation.sol#L218-L225

[N-05] Invalid/extraneous/optional function definitions in interface

There are 4 instances of this issue:

```
File: contracts/core/GolomTrader.sol

/// @audit withdraw(uint256) isn't defined with those arguments
33:        function withdraw(uint256 wad) external;
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L33

```
File: contracts/rewards/RewardDistributor.sol

/// @audit mint(address,uint256) isn't defined with those argume
24:        function mint(address account, uint256 amount) externa

/// @audit balanceOfNFTAt(uint256,uint256) isn't defined with th
26:        function balanceOfNFTAt(uint256 _tokenId, uint256 _t)

/// @audit deposit() isn't defined with those arguments in the s
28:        function deposit() external payable;
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L24

[N-06] Remove `include` for hardhat's console

There is 1 instance of this issue:

```
File: contracts/rewards/RewardDistributor.sol

9:    import 'hardhat/console.sol';
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L9

[N-07] Contract implements interface without extending the interface

Not extending the interface may lead to the wrong function signature being used, leading to unexpected behavior. If the interface is in fact being implemented, use the `override` keyword to indicate that fact

There is 1 instance of this issue:

```
File: contracts/vote-escrow/VoteEscrowCore.sol

/// @audit IERC721Enumerable.tokenOfOwnerByIndex()
275:  contract VoteEscrowCore is IERC721, IERC721Metadata {
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L275

[N-08] `require()` / `revert()` statements should have descriptive reason strings

There are 31 instances of this issue:

```
File: contracts/core/GolomTrader.sol

220:                 require(msg.sender == o.reservedAddress);

285             require(
286                 o.totalAmt * amount >
287                     (o.exchange.paymentAmt + o.prePayment.paym
288:            ); // cause bidder eth is paying for seller paymer

291:                 require(msg.sender == o.reservedAddress);

293:            require(o.orderType == 1);
```

```
295:            require(status == 3);

296:            require(amountRemaining >= amount);

313:            require(o.signer == msg.sender);

342:            require(o.totalAmt >= o.exchange.paymentAmt + o.pr

345:                require(msg.sender == o.reservedAddress);

347:            require(o.orderType == 2);

349:            require(status == 3);

350:            require(amountRemaining >= amount);
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L220

```
File: contracts/rewards/RewardDistributor.sol

88:             require(msg.sender == trader);

144:                require(epochs[index] < epoch);

158:                require(epochs[index] < epoch);
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L88

```
File: contracts/vote-escrow/VoteEscrowCore.sol

360:            require(_entered_state == _not_entered);

540:            require(_isApprovedOrOwner(_sender, _tokenId));

646:            require(owner != address(0));
```

```
648:         require(_approved != owner);

652:         require(senderIsOwner || senderIsApprovedForAll);

869:         require(msg.sender == voter);

874:         require(msg.sender == voter);

879:         require(msg.sender == voter);

884:         require(msg.sender == voter);

889:         require(msg.sender == voter);

895:         require(_from != _to);

896:         require(_isApprovedOrOwner(msg.sender, _from));

897:         require(_isApprovedOrOwner(msg.sender, _to));

927:         require(_value > 0); // dev: need non-zero value

944:         require(_value > 0); // dev: need non-zero value
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L360

```
File: contracts/vote-escrow/VoteEscrowDelegation.sol

245:         require(_isApprovedOrOwner(_sender, _tokenId));
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowDelegation.sol#L245

[N-09] `public` functions not called by the contract should be declared `external` instead

Contracts [are allowed]() to override their parents' functions and change the visibility from `external` to `public`.

There are 13 instances of this issue:

```
File: contracts/core/GolomTrader.sol

203       function fillAsk(
204           Order calldata o,
205           uint256 amount,
206           address referrer,
207           Payment calldata p,
208           address receiver
209:      ) public payable nonReentrant {

279       function fillBid(
280           Order calldata o,
281           uint256 amount,
282           address referrer,
283           Payment calldata p
284:      ) public nonReentrant {

312:      function cancelOrder(Order calldata o) public nonReent

334       function fillCriteriaBid(
335           Order calldata o,
336           uint256 amount,
337           uint256 tokenId,
338           bytes32[] calldata proof,
339           address referrer,
340           Payment calldata p
341:      ) public nonReentrant {
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L203-L209

```
File: contracts/rewards/RewardDistributor.sol

98:       function addFee(address[2] memory addr, uint256 fee) p

141:      function traderClaim(address addr, uint256[] memory ep
155:     function exchangeClaim(address addr, uint256[] memory

172:     function multiStakerClaim(uint256[] memory tokenids, u

215      function stakerRewards(uint256 tokenid) public view re
216              uint256,
217              uint256,
218:             uint256[] memory

254      function traderRewards(address addr) public view retur
255:             uint256

269      function exchangeRewards(address addr) public view ret
270:             uint256
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L98

```
File: contracts/vote-escrow/TokenUriHelper.sol

66       function _tokenURI(
67           uint256 _tokenId,
68           uint256 _balanceOf,
69           uint256 _locked_end,
70           uint256 _value
71:      ) public pure returns (string memory output) {
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/TokenUriHelper.sol#L66-L71

```
File: contracts/vote-escrow/VoteEscrowDelegation.sol

185:     function getPriorVotes(uint256 tokenId, uint256 blockN
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowDelegation.sol#L185

[N-10] Non-assembly method available

`assembly{ id := chainid() }` => `uint256 id = block.chainid`, `assembly { size := extcodesize() }` => `uint256 size = address().code.length`
There are some automated tools that will flag a project as having higher complexity if there is inline-assembly, so it's best to avoid using it where it's not necessary.

There are 2 instances of this issue:

```
File: contracts/core/GolomTrader.sol

98:                 chainId := chainid()
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L98

```
File: contracts/vote-escrow/VoteEscrowCore.sol

577:                size := extcodesize(account)
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L577

[N-11] `constant`s should be defined rather than using magic numbers

Even assembly can benefit from using readable constants instead of hex/numeric literals

There are 49 instances of this issue:

```
File: contracts/core/GolomTrader.sol
```

```
/// @audit 50
/// @audit 10000
212:                o.totalAmt >= o.exchange.paymentAmt + o.prePay

/// @audit 3
226:            require(status == 3, 'order not valid');

/// @audit 50
/// @audit 10000
242:            payEther(((o.totalAmt * 50) / 10000) * amount, adc

/// @audit 50
254:                        (o.totalAmt * 50) /

/// @audit 10000
255:                        10000 -

/// @audit 50
/// @audit 10000
263:                    (o.totalAmt - (o.totalAmt * 50) / 10000 -

/// @audit 50
/// @audit 10000
269:            distributor.addFee([o.signer, o.exchange.paymentAc

/// @audit 3
295:            require(status == 3);

/// @audit 3
349:            require(status == 3);

/// @audit 50
/// @audit 10000
381:            uint256 protocolfee = ((o.totalAmt * 50) / 10000)

/// @audit 0x00
436:                mstore(0x00, a)

/// @audit 0x20
437:                mstore(0x20, b)

/// @audit 0x00
/// @audit 0x40
438:                value := keccak256(0x00, 0x40)
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L212

```
File: contracts/governance/GolomToken.sol

/// @audit 150_000_000
/// @audit 1e18
44:             _mint(_airdrop, 150_000_000 * 1e18);

/// @audit 62_500_000
/// @audit 1e18
52:             _mint(_rewardDistributor, 62_500_000 * 1e18);
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/governance/GolomToken.sol#L44

```
File: contracts/rewards/RewardDistributor.sol

/// @audit 1659211200
84:             startTime = 1659211200;

/// @audit 1000000000
/// @audit 18
100:            if (rewardToken.totalSupply() > 1000000000 * 10**1

/// @audit 67
/// @audit 100
120:                rewardTrader[epoch] = ((tokenToEmit - stakerRe

/// @audit 33
/// @audit 100
121:                rewardExchange[epoch] = ((tokenToEmit - staker
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L84

```
File: contracts/vote-escrow/TokenUriHelper.sol

/// @audit 4
/// @audit 3
17:            uint256 encodedLen = 4 * ((len + 2) / 3);

/// @audit 32
20:            bytes memory result = new bytes(encodedLen + 32);

/// @audit 0xffffff
34:                    let input := and(mload(add(data, i)), 0xf

/// @audit 0x3F
36:                    let out := mload(add(tablePtr, and(shr(18,

/// @audit 0x3F
/// @audit 0xFF
38:                    out := add(out, and(mload(add(tablePtr, ar

/// @audit 0x3F
/// @audit 0xFF
40:                    out := add(out, and(mload(add(tablePtr, ar

/// @audit 0x3F
/// @audit 0xFF
42:                    out := add(out, and(mload(add(tablePtr, ar

/// @audit 0x3d3d
52:                    mstore(sub(resultPtr, 2), shl(240, 0x3d3d)

/// @audit 0x3d
55:                    mstore(sub(resultPtr, 1), shl(248, 0x3d))

/// @audit 48
144:               buffer[digits] = bytes1(uint8(48 + uint256(val
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/TokenUriHelper.sol#L17

```
File: contracts/vote-escrow/VoteEscrowCore.sol

/// @audit 255
```

```
745:                    for (uint256 i = 0; i < 255; ++i) {

/// @audit 128
1044:           for (uint256 i = 0; i < 128; ++i) {

/// @audit 128
1115:           for (uint256 i = 0; i < 128; ++i) {

/// @audit 255
1167:           for (uint256 i = 0; i < 255; ++i) {
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L745

```
File: contracts/vote-escrow/VoteEscrowDelegation.sol

/// @audit 500
99:             require(_delegatedTokenIds.length < 500, 'VVDelega
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowDelegation.sol#L99

[N-12] Numeric values having to do with time should use time units for readability

There are [units](#) for seconds, minutes, hours, days, and weeks, and since they're defined, they should be used

There are 5 instances of this issue:

```
File: contracts/rewards/RewardDistributor.sol

/// @audit 600000
48:         uint256 constant dailyEmission = 600000 * 10**18;

/// @audit 60
/// @audit 60
```

```
57:        uint256 constant secsInDay = 24 * 60 * 60;
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L48

```
File: contracts/vote-escrow/VoteEscrowCore.sol

/// @audit 86400
296:        uint256 internal constant MAXTIME = 4 * 365 * 86400;

/// @audit 86400
297:        int128 internal constant iMAXTIME = 4 * 365 * 86400;
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L296

[N-13] Large multiples of ten should use scientific notation (e.g. `1e6`) rather than decimal literals (e.g. `1000000`), for readability

There are 2 instances of this issue:

```
File: contracts/rewards/RewardDistributor.sol

100:            if (rewardToken.totalSupply() > 1000000000 * 10**1
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L100

```
File: contracts/vote-escrow/VoteEscrowCore.sol

308:        mapping(uint256 => Point[1000000000]) public user_poir
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L308

[N-14] Use a more recent version of solidity

Use a solidity version of at least 0.8.12 to get `string.concat()` to be used instead of `abi.encodePacked(<str>,<str>)`

There are 2 instances of this issue:

```
File: contracts/core/GolomTrader.sol

3:    pragma solidity 0.8.11;
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L3

```
File: contracts/vote-escrow/TokenUriHelper.sol

3:    pragma solidity 0.8.11;
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/TokenUriHelper.sol#L3

[N-15] Use scientific notation (e.g. `1e18`) rather than exponentiation (e.g. `10**18`)

While the compiler knows to optimize away the exponentiation, it's still better coding practice to use idioms that do not require compiler optimization, if they exist

There are 2 instances of this issue:

```
File: contracts/rewards/RewardDistributor.sol
```

```
48:        uint256 constant dailyEmission = 600000 * 10**18;

100:            if (rewardToken.totalSupply() > 1000000000 * 10**1
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L48

[N-16] Lines are too long

Usually lines in source code are limited to 80 characters. Today's screens are much larger so it's reasonable to stretch this in some cases. Since the files will most likely reside in GitHub, and GitHub starts using a scroll bar in all cases when the length is over 164 characters, the lines below should be split when they reach that length

There are 8 instances of this issue:

```
File: contracts/core/GolomTrader.sol

132:                         'order(address collection,uint256

329:        ///      to send ether to that address on filling the
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L132

```
File: contracts/rewards/RewardDistributor.sol

126:                       epochTotalFee[0] =  address(this).bala
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L126

```
File: contracts/vote-escrow/TokenUriHelper.sol
```

```
72:             output = '<svg xmlns="http://www.w3.org/2000/svg"

78:                     '</text><text y="318px" x="54px" fill="whi

86:                     '</text><text y="248px" x="54px" fill="whi

94:                     '</text><text y="391px" x="54px" fill="whi

102:                    '</text><mask id="mask0_2_190" style="mask
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/TokenUriHelper.sol#L72

[N-17] Inconsistent method of specifying a floating pragma

Some files use `>=`, some use `^`. The instances below are examples of the method that has the fewest instances for a specific version. Note that using `>=` without also specifying `<=` will lead to failures to compile, or external project incompatability, when the major version changes and there are breaking-changes, so `^` should be preferred regardless of the instance counts

There is 1 instance of this issue:

```
File: contracts/governance/GolomToken.sol

2:    pragma solidity ^0.8.11;
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/governance/GolomToken.sol#L2

[N-18] Variable names that consist of all capital letters should be reserved for `constant`/`immutable` variables

If the variable needs to be different based on which class it comes from, a `view`/`pure` *function* should be used instead (e.g. like this).

There are 2 instances of this issue:

```
File: contracts/core/GolomTrader.sol

45:       ERC20 WETH = ERC20(0xC02aaA39b223FE8D0A0e5C4F27eAD9083
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L45

```
File: contracts/vote-escrow/VoteEscrowDelegation.sol

50:       uint256 public MIN_VOTING_POWER_REQUIRED = 0;
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowDelegation.sol#L50

[N-19] Non-library/interface files should use fixed compiler versions, not floating ones

There is 1 instance of this issue:

```
File: contracts/governance/GolomToken.sol

2:    pragma solidity ^0.8.11;
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/governance/GolomToken.sol#L2

[N-20] File does not contain an SPDX Identifier

There is 1 instance of this issue:

```
File: contracts/vote-escrow/TokenUriHelper.sol

0:    /// [MIT License]
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/TokenUriHelper.sol#L0

[N-21] Event is missing `indexed` fields

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it’s not necessarily best to index the maximum allowed per event (threefields). Each `event` should use three `indexed` fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

There are 7 instances of this issue:

```
File: contracts/core/GolomTrader.sol

79:        event NonceIncremented(address indexed maker, uint256
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L79

```
File: contracts/rewards/RewardDistributor.sol

70:        event NewEpoch(uint256 indexed epochNo, uint256 tokenN
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L70

```
File: contracts/vote-escrow/VoteEscrowCore.sol

67:       event ApprovalForAll(address indexed owner, address i

284       event Deposit(
285           address indexed provider,
286           uint256 tokenId,
287           uint256 value,
288           uint256 indexed locktime,
289           DepositType deposit_type,
290           uint256 ts
291:      );

292:      event Withdraw(address indexed provider, uint256 toke

293:      event Supply(uint256 prevSupply, uint256 supply);
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowCore.sol#L67

```
File: contracts/vote-escrow/VoteEscrowDelegation.sol

29:       event DelegateVotesChanged(address indexed delegate,
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowDelegation.sol#L29

[N-22] Duplicated `require()` / `revert()` checks should be refactored to a modifier or function

The compiler will inline the function, which will avoid `JUMP` instructions usually associated with functions

There are 14 instances of this issue:

```
File: contracts/core/GolomTrader.sol
```

```
291:                require(msg.sender == o.reservedAddress);

299:                require(amount == 1, 'only 1 erc721 at 1 time'

349:            require(status == 3);

350:            require(amountRemaining >= amount);
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/core/GolomTrader.sol#L291

```
File: contracts/rewards/RewardDistributor.sol

158:                require(epochs[index] < epoch);

220:            require(address(ve) != address(0), ' VE not added
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/rewards/RewardDistributor.sol#L158

```
File: contracts/vote-escrow/VoteEscrowCore.sol

1008:           require(attachments[_tokenId] == 0 && !voted[_toke

874:            require(msg.sender == voter);

944:            require(_value > 0); // dev: need non-zero value

982:            require(_locked.amount > 0, 'No existing lock four

983:            require(_locked.end > block.timestamp, 'Cannot adc

999:            require(unlock_time <= block.timestamp + MAXTIME,
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-

escrow/VoteEscrowCore.sol#L1008

```
File: contracts/vote-escrow/VoteEscrowDelegation.sol

211:            require(ownerOf(tokenId) == msg.sender, 'VEDelegat

186:            require(blockNumber < block.number, 'VEDelegation:
```

https://github.com/code-423n4/2022-07-golom/blob/e5efa8f9d6dda92a90b8b2c4902320acf0c26816/contracts/vote-escrow/VoteEscrowDelegation.sol#L211

Gas Optimizations

For this contest, 92 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **JohnSmith** received the top score from the judge.

The following wardens also submitted reports: __141345__, _Adam, 0x1f8b, 0xA5DF, 0xDjango, 0xKitsune, 0xLovesleep, 0xmatt, 0xNazgul, 0xsam, 0xSmartContract, ajtra, ak1, apostle0x01, asutorufos, async, Aymen0909, benbaessler, Bnke0x0, brgltd, c3phas, carlitox477, Chandr, Chinmay, CodingNameKiki, cRat1st0s, CRYP70, Deivitto, delfin454000, djxploit, Dravee, durianSausage, ElKu, ellahi, erictee, fatherOfBlocks, Fitraldys, Funen, GalloDaSballo, gerdusx, gogo, Green, hyh, IllIIII, jayfromthe13th, jayphbee, JC, Jmaxmanblue, joestakey, Junnon, kaden, Kaiziron, Kenshin, kenzo, Krow10, kyteg, ladboy233, lucacez, m_Rassska, Maxime, mics, Migue, MiloTruck, minhquanym, Noah3o6, NoamYakov, oyc_109, pfapostol, Randyyy, rbserver, reassor, RedOneN, ReyAdmirado, rfa, robee, Rohan16, rokinot, Rolezn, Ruhum, sach1r0, saian, samruna, sashik_eth, simon135, Sm4rty, StyxRave, supernova, tofunmi, Tomio, TomJ, *and* zuhaibmohd.

	Issue	Instances
1	++variable costs less gas than variable++	12
2	Access mappings directly rather than using accessor functions	6
3	Cheaper input valdiations should come before expensive operations	3

	Issue	Instances
4	Help the optimizer by saving a storage variable's reference	1
5	The result of external function calls should be cached rather than re-calling the function	4
6	Amounts should be checked for `0` before calling a transfer	10
7	`public` functions to `external`	8
8	Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas	4
9	Default value initialization	2
10	Multiplication/division by 2 should use bit shifting	3
11	Splitting `require()` statements that use `&&` saves gas	4
12	`abi.encode()` is less efficient than `abi.encodePacked()`	4
13	`struct Order` : can be more tighly packed	1
14	State variables can be packed into fewer storage slots	8
15	`x += y` costs more gas than `x = x + y` for state variables	2
16	`internal` and `private` functions only called once can be inlined to save gas	10
17	Unchecked arithmetic	50
18	Using `bool` s for storage incurs overhead	5
19	Using `> 0` costs more gas than `!= 0` when used on a `uint` in a `require()` and `assert` statement	3
20	Usage of `uints` / `ints` smaller than 32 bytes (256 bits) incurs overhead	11
21	Use custom errors rather than `revert()` / `require()` strings to save gas	76
22	Multiple `address` mappings can be combined into a single `mapping` of an `address` to a `struct` , where appropriate	5
23	Functions guaranteed to revert when called by normal users can be marked `payable`	12
24	Do not calculate constants	4
25	`<array>.length` should not be looked up in every loop of a for-loop	8

	Issue	Instances
26	Copying struct to memory can be more expensive than just reading from storagexxx	3
27	`require()` / `revert()` strings longer than 32 bytes cost extra gas	8
28	Remove or replace unused variables	2
29	State variables only set in the constructor should be declared `immutable`	3
30	State variables with values known at compile time should be constants	2
31	State variables should be cached in stack variables rather than re-reading them from storage	14
32	Using `private` rather than `public` for constants, saves gas	4
33	Remove unreachable code	1
34	No need to evaluate all expressions to know if one of them is true	2
35	No need to read `tokenId` second time	1
36	`last_point` value rewritten right after initialization	1
37	Wasted gas on copying a struct	1
38	Remove duplicate code	1
39	No need for mapping `supportedInterfaces` to exist	1
40	Same calculation twice	1
41	Obsolete constants	2
42	Variable `end` can be of type `uint128`	1

[G-01] ++variable costs less gas than variable++, especially when it’s used in for-loops (—variable/variable— too)

Prefix increments are cheaper than postfix increments.
Saves 6 gas *PER LOOP*
There are 12 instances of this issue:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L143

```
File: contracts/rewards/RewardDistributor.sol
143:    for (uint256 index = 0; index < epochs.length; index++)

157:    for (uint256 index = 0; index < epochs.length; index++)

180:    for (uint256 tindex = 0; tindex < tokenids.length; tinde

183:    for (uint256 index = 0; index < epochs.length; index++)

226:    for (uint256 index = 0; index < epoch; index++) {

258:    for (uint256 index = 0; index < epoch; index++) {

273:    for (uint256 index = 0; index < epoch; index++) {
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/core/GolomTrader.sol#L415

```
File:   contracts/core/GolomTrader.sol
415:    for (uint256 i = 0; i < proof.length; i++) {
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowDelegation.sol#L171

```
File:   contracts/vote-escrow/VoteEscrowDelegation.sol
171:    for (uint256 index = 0; index < delegated.length; index+

189:    for (uint256 index = 0; index < delegatednft.length; inc
199:    for (uint256 i; i < _array.length; i++) {
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/TokenUriHelper.sol#L138

```
File:   contracts/vote-escrow/TokenUriHelper.sol
138:    digits++;
```

Mitigation

Change `i++` to `++i`

[G-02] Access mappings directly rather than using accessor functions

Saves having to do two `JUMP` instructions, along with stack setup

There are 6 instances of this issue:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowDelegation.sol#L72

```
File:   contracts/vote-escrow/VoteEscrowDelegation.sol
72:             require(ownerOf(tokenId) == msg.sender, 'VEDeleg

211:  require(ownerOf(tokenId) == msg.sender, 'VEDelegation: Not
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L1229

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
1229:   address owner = ownerOf(_tokenId);

406:    return _balance(_owner);
```

```
453:    uint256 current_count = _balance(_to);

464:    uint256 current_count = _balance(_from) - 1;
```

Mitigation

Istead of `ownerOf(tokenId)` use `idToOwner[tokenId]`

[G-03] Cheaper input valdiations should come before expensive operations

Check @audit comment for details

There are 3 instances of this issue:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L925

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
925:        LockedBalance memory _locked = locked[_tokenId];
926:                                    //@audit this check shou
927:        require(_value > 0); // dev: need non-zero value

942:        uint256 unlock_time = ((block.timestamp + _lock_dura
943:                                    //@audit this check shou
944:        require(_value > 0); // dev: need non-zero value

977:        assert(_isApprovedOrOwner(msg.sender, _tokenId));
978:
979:        LockedBalance memory _locked = locked[_tokenId];
980:                                    //@audit this check shou
981:        assert(_value > 0); // dev: need non-zero value
```

[G-04] Help the optimizer by saving a storage variable’s reference instead of repeatedly fetching it

To help the optimizer, declare a `storage` type variable and use it instead of repeatedly fetching the reference in a map or an array. The effect can be quite

significant.
As an example, instead of repeatedly calling `someMap[someIndex]` , save its reference like this: `SomeStruct storage someStruct = someMap[someIndex]` and use it.
Instances include:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowDelegation.sol#L138

```
File:   contracts/vote-escrow/VoteEscrowDelegation.sol
138:         if (checkpoints[nftId][nCheckpoints - 1].fromBlock <
139:             return checkpoints[nftId][nCheckpoints - 1].dele
```

Declare `storage` variable and use it:

```
Checkpoint storage checkpoint = checkpoints[nftId][nCheckpoints
if (checkpoint.fromBlock <= blockNumber) {
    return checkpoint.delegatedTokenIds;
}
```

[G-05] The result of external function calls should be cached rather than re-calling the function

There is no need to call another contract functions multiple times to get same value, returned value should be cached in a variable; The instances below point to the second+ call of the function within a single function:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L100

```
File:   contracts/rewards/RewardDistributor.sol
100: if (rewardToken.totalSupply() > 1000000000 * 10**18) {

112:             uint256 tokenToEmit = (dailyEmission * (rewardTo
113:                     rewardToken.totalSupply();
```

```
114:             uint256 stakerReward = (tokenToEmit * rewardToke

239:        (rewardStaker[index] * ve.balanceOfAtNFT(tokenid, epoc
240:        ve.totalSupplyAt(epochBeginTime[index]);

244:         ve.balanceOfAtNFT(tokenid, epochBeginTime[index])) /
245:       ve.totalSupplyAt(epochBeginTime[index]);
```

`rewardToken.totalSupply()` should be cached. `ve.balanceOfAtNFT(tokenid, epochBeginTime[index]))` should be cached.

`ve.totalSupplyAt(epochBeginTime[index])` should be cached.

[G-06] Amounts should be checked for `0` before calling a transfer

Checking non-zero transfer values can avoid an expensive external call and save gas.
While this is done at some places, it's not consistently done in the solution.
I suggest adding a non-zero-value check here:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L151

```
File:   contracts/rewards/RewardDistributor.sol
151: rewardToken.transfer(addr, reward);

165: rewardToken.transfer(addr, reward);

208:  rewardToken.transfer(tokenowner, reward);

209:  weth.transfer(tokenowner, rewardEth);
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/core/GolomTrader.sol#L238

```
File:   contracts/core/GolomTrader.sol
```

```
238: ERC1155(o.collection).safeTransferFrom(o.signer, receiver,

304: nftcontract.safeTransferFrom(msg.sender, o.signer, o.token

364: nftcontract.safeTransferFrom(msg.sender, o.signer, tokenId,

382: WETH.transferFrom(o.signer, address(this), o.totalAmt * am

383: WETH.withdraw(o.totalAmt * amount);
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L1023

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
1023: assert(IERC20(token).transfer(msg.sender, value));
```

[G-07] `public` functions to `external`

External call cost is less expensive than of public functions.
Contracts [are allowed](https://github.com/ethereum/solidity/issues/11867) to override their parents' functions and change the visibility from `external` to `public`.
The following functions could be set `external` to save gas and improve code quality:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L98

```
File:   contracts/rewards/RewardDistributor.sol
98:   function addFee(address[2] memory addr, uint256 fee) publi

141: function traderClaim(address addr, uint256[] memory epochs)

155: function exchangeClaim(address addr, uint256[] memory epoch

172: function multiStakerClaim(uint256[] memory tokenids, uint2
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/core/GolomTrader.sol#L203

```
File:   contracts/core/GolomTrader.sol
203: function fillAsk(
204:         Order calldata o,
205:         uint256 amount,
206:         address referrer,
207:         Payment calldata p,
208:         address receiver
209:     ) public payable nonReentrant {

279: function fillBid(
280:         Order calldata o,
281:         uint256 amount,
282:         address referrer,
283:         Payment calldata p
284:     ) public nonReentrant {

312: function cancelOrder(Order calldata o) public nonReentrant

334: function fillCriteriaBid(
335:         Order calldata o,
336:         uint256 amount,
337:         uint256 tokenId,
338:         bytes32[] calldata proof,
339:         address referrer,
340:         Payment calldata p
341:     ) public nonReentrant {
```

[G-08] Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas

If you choose to make suggested above `public` functions as `external`, to continue gas optimizaton we can use `calldata` function arguments instead of `memory`.

When a function with a `memory` array is called externally, the `abi.decode()` step has to use a for-loop to copy each index of the `calldata` to the `memory` index. Each iteration of this for-loop costs at least 60 gas (i.e. `60 *`

`<mem_array>.length`). Using `calldata` directly, obliviates the need for such a loop in the contract code and runtime execution. Structs have the same overhead as an array of length one.
Instances include:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L98

```
File:   contracts/rewards/RewardDistributor.sol
98:  function addFee(address[2] memory addr, uint256 fee) public

141: function traderClaim(address addr, uint256[] memory epochs)

155: function exchangeClaim(address addr, uint256[] memory epoch

172: function multiStakerClaim(uint256[] memory tokenids, uint25
```

[G-09] Default value initialization

Problem

If a variable is not set/initialized, it is assumed to have the default value (`0` , `false` , `0x0` etc depending on the data type).
Explicitly initializing it with its default value is an anti-pattern and wastes gas.
Instances include:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L45

```
File:   contracts/rewards/RewardDistributor.sol
45:  uint256 public epoch = 0;
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowDelegation.sol#L50

```
File:   contracts/vote-escrow/VoteEscrowDelegation.sol
uint256 public MIN_VOTING_POWER_REQUIRED = 0;
```

Proof of Concept

In case of `RewardDistributor`

`hardhat-gas-reporter` results

before fix:
| Deployments |||| |-|:-:|:-:|:-:| | Contract | min | max | avg | | GolomTrader | 2379507 | 2379543 | 2379537 | after fix: | Deployments |||| |-|:-:|:-:|:-:| | Contract | min | max | avg | | GolomTrader | 2377233 | 2377269 | 2377263 |

Mitigation

Remove explicit initialization for default values.

[G-10] Multiplication/division by 2 should use bit shifting

`<x> * 2` is equivalent to `<x> << 1` and `<x> / 2` is the same as `<x> >> 1`. The `MUL` and `DIV` opcodes cost 5 gas, whereas `SHL` and `SHR` only cost 3 gas.

There are 3 instances of this issue:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowDelegation.sol#L150

```
File:   contracts/vote-escrow/VoteEscrowDelegation.sol
150: uint256 center = upper - (upper - lower) / 2; // ceil, avoi
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L1049

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
1049: uint256 _mid = (_min + _max + 1) / 2;
```

```
1120: uint256 _mid = (_min + _max + 1) / 2;
```

[G-11] Splitting `require()` statements that use `&&` saves gas

Instead of using the `&&` operator in a single require statement to check multiple conditions, I suggest using multiple require statements with 1 condition per require statement (saving 3 gas per `&`).
See this issue which describes the fact that there is a larger deployment gas cost, but with enough runtime calls, the change ends up being cheaper.

There are 4 instances of this issue:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowDelegation.sol#L239

```
File:   contracts/vote-escrow/VoteEscrowDelegation.sol
239: require(attachments[_tokenId] == 0 && !voted[_tokenId], 'at
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L538

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
538: require(attachments[_tokenId] == 0 && !voted[_tokenId], 'at
894: require(attachments[_from] == 0 && !voted[_from], 'attachec
1008: require(attachments[_tokenId] == 0 && !voted[_tokenId], 'a
```

[G-12] `abi.encode()` is less efficient than `abi.encodePacked()`

Consider changing it if possible.
There are 4 instances of this issue:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/core/Gol

omTrader.sol#L102

```
File:   contracts/core/GolomTrader.sol
102:  abi.encode(
115:    abi.encode(
130:    abi.encode(
414:  bytes32 computedHash = keccak256(abi.encode(leaf));
```

[G-13] `struct Order` : can be more tighly packed

Each slot saved can avoid an extra Gsset (20000 gas) for the first setting of the struct. Subsequent reads as well as writes have smaller gas savings.

Before:
https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/core/GolomTrader.sol#L47

```
File:   contracts/core/GolomTrader.sol
47:     struct Order {
        address collection; // NFT contract address
        uint256 tokenId; // order for which tokenId of the colle
        address signer; // maker of order address
        uint256 orderType; // 0 if selling nft for eth , 1 if of
        uint256 totalAmt; // price value of the trade // total a
        Payment exchange; // payment agreed by maker of the orde
        Payment prePayment; // another payment , can be used for
        bool isERC721; // standard of the collection , if 721 th
        uint256 tokenAmt; // token amt useful if standard is 115
        uint256 refererrAmt; // amt to pay to the address that h
        bytes32 root; // A merkle root derived from each valid t
        address reservedAddress; // if not address(0) , only thi
        uint256 nonce; // nonce of order usefull for cancelling
        uint256 deadline; // timestamp till order is valid epoch
        uint8 v;
        bytes32 r;
        bytes32 s;
    }
```

Methods					
Contract	Method	min	max	avg	#calls
GolomTrader	fillAsk	238167	241974	241425	7
GolomTrader	setDistributor	46281	70449	47432	21

Deployments	
Contract	avg
GolomTrader	2029204

After:

```
        struct Order {
            address collection; // NFT contract address
            uint256 tokenId; // order for which tokenId of the colle
            address signer; // maker of order address
            uint256 orderType; // 0 if selling nft for eth , 1 if of
            uint256 totalAmt; // price value of the trade // total a
            Payment exchange; // payment agreed by maker of the orde
            Payment prePayment; // another payment , can be used for
            uint256 tokenAmt; // token amt useful if standard is 115
            uint256 refererrAmt; // amt to pay to the address that h
            bytes32 root; // A merkle root derived from each valid t
            address reservedAddress; // if not address(0) , only thi
            uint256 nonce; // nonce of order usefull for cancelling
            uint256 deadline; // timestamp till order is valid epoch
            bool isERC721; // standard of the collection , if 721 th
            uint8 v;
            bytes32 r;
            bytes32 s;
        }
```

Methods					
Contract	Method	min	max	avg	#calls
GolomTrader	fillAsk	238153	241948	241394	7
GolomTrader	setDistributor	46259	70427	47410	21

Deployments	
Contract	avg
GolomTrader	2013782

[G-14] State variables can be packed into fewer storage slots

These state variables can be packed together to use less storage slots:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L297

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
297: int128 internal constant iMAXTIME = 4 * 365 * 86400;

320: uint8 public constant decimals = 18;

347: bytes4 internal constant ERC165_INTERFACE_ID = 0x01ffc9a7;
350: bytes4 internal constant ERC721_INTERFACE_ID = 0x80ac58cd;
353: bytes4 internal constant ERC721_METADATA_INTERFACE_ID = 0x5

356: uint8 internal constant _not_entered = 1;
357: uint8 internal constant _entered = 2;
358: uint8 internal _entered_state = 1;
```

[G-15] `x += y` costs more gas than `x = x + y` for state variables

`x += y` costs more than `x = x + y`

same as `x -= y`

Instances include:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L499

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
```

```
499: ownerToNFTokenCount[_to] += 1;

512: ownerToNFTokenCount[_from] -= 1;
```

Mitigation

Replace `x += y` and `x -= y` with `x = x + y` and `x = x - y`.

[G-16] `internal` and `private` functions only called once can be inlined to save gas

Not inlining costs **20 to 40 gas** because of two extra `JUMP` instructions and additional stack operations needed for function calls.

There are 10 instances of this issue:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowDelegation.sol#L116

```
File:   contracts/vote-escrow/VoteEscrowDelegation.sol
116: function _getCurrentDelegated(uint256 tokenId) internal vie
{
129: function _getPriorDelegated(uint256 nftId, uint256 blockNum

198: function removeElement(uint256[] storage _array, uint256 _e
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L452

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
452: function _addTokenToOwnerList(address _to, uint256 _tokenIc

462: function _removeTokenFromOwnerList(address _from, uint256 _

571: function _isContract(address account) internal view returns
```

```
677: function _mint(address _to, uint256 _tokenId) internal retu

1107: function _balanceOfAtNFT(uint256 _tokenId, uint256 _block)
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/governance/GolomToken.sol#L123

```
File:   contracts/governance/GolomToken.sol
123: function _hashOrder(Order calldata o) private pure returns

127: function _hashOrderinternal(Order calldata o, uint256[2] me
```

[G-17] Unchecked arithmetic

The default "checked" behavior costs more gas when adding/diving/multiplying, because under-the-hood those checks are implemented as a series of opcodes that, prior to performing the actual arithmetic, check for under/overflow and revert if it is detected.
If it can statically be determined there is no possible way for your arithmetic to under/overflow (such as a condition in an if statement), surrounding the arithmetic in an unchecked block will save gas.
Instances include:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/governance/GolomToken.sol#L60

```
File:   contracts/governance/GolomToken.sol
minterEnableDate = block.timestamp + 1 days;
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L106

```
File:   contracts/rewards/RewardDistributor.sol
```

```
106: if (block.timestamp > startTime + (epoch) * secsInDay) {

112:  uint256 tokenToEmit = (dailyEmission * (rewardToken.totalS
113:    rewardToken.totalSupply();
114:  uint256 stakerReward = (tokenToEmit * rewardToken.balance(

118: epoch = epoch + 1;

143: for (uint256 index = 0; index < epochs.length; index++) {

157: for (uint256 index = 0; index < epochs.length; index++) {

180: for (uint256 tindex = 0; tindex < tokenids.length; tindex++

183: for (uint256 index = 0; index < epochs.length; index++) {

258: for (uint256 index = 0; index < epoch; index++) {

273: for (uint256 index = 0; index < epoch; index++) {

286: traderEnableDate = block.timestamp + 1 days;

302: voteEscrowEnableDate = block.timestamp + 1 days;
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/core/GolomTrader.sol#L324

```
File:   contracts/core/GolomTrader.sol
324: uint256 newNonce = ++nonces[msg.sender];

415: for (uint256 i = 0; i < proof.length; i++) {

449: distributorEnableDate = block.timestamp + 1 days;
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowDelegation.sol#L107

```
File:   contracts/vote-escrow/VoteEscrowDelegation.sol
107: numCheckpoints[toTokenId] = nCheckpoints + 1;
```

```
119: return nCheckpoints > 0 ? checkpoints[tokenId][nCheckpoints

138:     if (checkpoints[nftId][nCheckpoints - 1].fromBlock <= bl
139:         return checkpoints[nftId][nCheckpoints - 1].delegate

148: uint256 upper = nCheckpoints - 1;

150: uint256 center = upper - (upper - lower) / 2;

157: upper = center - 1;

171: for (uint256 index = 0; index < delegated.length; index++)
172:         votes = votes + this.balanceOfNFT(delegated[index]);

189:     for (uint256 index = 0; index < delegatednft.length; inc
190:         votes = votes + this.balanceOfAtNFT(delegatednft[inc

199: for (uint256 i; i < _array.length; i++) {

201: _array[i] = _array[_array.length - 1];
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L499

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
499: ownerToNFTokenCount[_to] += 1;

512: ownerToNFTokenCount[_from] -= 1;

745: for (uint256 i = 0; i < 255; ++i) {

748: t_i += WEEK;

768: _epoch += 1;

994: uint256 unlock_time = ((block.timestamp + _lock_duration) /

999: require(unlock_time <= block.timestamp + MAXTIME, 'Voting l

1044: for (uint256 i = 0; i < 128; ++i) {
```

```
1049: uint256 _mid = (_min + _max + 1) / 2;

1053: _max = _mid - 1;

1115: for (uint256 i = 0; i < 128; ++i) {

1120: uint256 _mid = (_min + _max + 1) / 2;

1124: _max = _mid - 1;

1167: for (uint256 i = 0; i < 255; ++i) {

1213:  Point memory point_next = point_history[target_epoch + 1]
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/TokenUriHelper.sol#L138

```
File:   contracts/vote-escrow/TokenUriHelper.sol
138:            digits++;
139:            temp /= 10;

143:            digits -= 1;
144:            buffer[digits] = bytes1(uint8(48 + uint256(value
145:            value /= 10;
```

Mitigation

Place the arithmetic operations in an `unchecked` block

```
        for (uint i; i < length;) {
                ...
                unchecked{ ++i; }
        }
```

[G-18] Using `bool` s for storage incurs overhead

```
    // Booleans are more expensive than uint256 or any type that
```

```
        // word because each write operation emits an extra SLOAD tc
        // slot's contents, replace the bits taken up by the boolear
        // back. This is the compiler's defense against contract upc
        // pointer aliasing, and it cannot be disabled.
```

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/58f635312aa21f947cae5f8578638a85aa2519f5/contracts/security/ReentrancyGuard.sol#L23-L27
Use `uint256(1)` and `uint256(2)` for true/false to avoid a Gwarmaccess (100 gas) for the extra SLOAD, and to avoid Gsset (**20000 gas**) when changing from 'false' to 'true', after having been 'true' in the past.

There are 5 instances of this issue:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/governance/GolomToken.sol#L20

```
File:   contracts/governance/GolomToken.sol
20:    bool public isAirdropMinted;
21:    bool public isGenesisRewardMinted;
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L314

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
314: mapping(uint256 => bool) public voted;

341: mapping(address => mapping(address => bool)) internal owner

344: mapping(bytes4 => bool) internal supportedInterfaces;
```

[G-19] Using `> 0` costs more gas than `!= 0` when used on a `uint` in a `require()` and `assert` statement

`> 0` is less efficient than `!= 0` for unsigned integers.
`!= 0` costs less gas compared to `> 0` for unsigned integers in `require` and `assert` statements with the optimizer enabled (6 gas).
There are 3 instances of this issue:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L927

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
927: require(_value > 0); // dev: need non-zero value

944: require(_value > 0); // dev: need non-zero value

981: assert(_value > 0); // dev: need non-zero value
```

Mitigation

Replace `> 0` with `!= 0`
Or update soldity compiler to >=0.8.13

[G-20] Usage of `uints` / `ints` smaller than 32 bytes (256 bits) incurs overhead

> When using elements that are smaller than 32 bytes, your contract's gas usage may be higher. This is because the EVM operates on 32 bytes at a time. Therefore, if the element is smaller than that, the EVM must use more operations in order to reduce the size of the element from 32 bytes to the desired size.
> It is only beneficial to use reduced-size arguments if you are dealing with storage values because the compiler will pack multiple elements into one storage slot, and thus, combine multiple reads or writes into a single operation. When dealing with function arguments or memory values, there is no inherent benefit because the compiler does not pack these values.

https://docs.soliditylang.org/en/v0.8.11/internals/layout_in_storage.html

Use a larger size then downcast where needed.
Instances include:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L271

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
271: int128 amount; //@audit no storage slot saved

297: int128 internal constant iMAXTIME = 4 * 365 * 86400;

311: mapping(uint256 => int128) public slope_changes; // time ->

320: uint8 public constant decimals = 18;

356:  uint8 internal constant _not_entered = 1;
357:    uint8 internal constant _entered = 2;
358:  uint8 internal _entered_state = 1;

697:  int128 old_dslope = 0;
698:  int128 new_dslope = 0;

749:  int128 d_slope = 0;

1169: int128 d_slope = 0;
```

[G-21] Use custom errors rather than `revert()` / `require()` strings to save gas

Custom errors are available from solidity version 0.8.4. Custom errors save **~50 gas** each time they're hit by avoiding having to allocate and store the revert string. Not defining the strings also saves deployment gas.

There are 76 instances of this issue:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/governance/GolomToken.sol#L24

```
File:   contracts/governance/GolomToken.sol
24: require(msg.sender == minter, 'GolomToken: only reward distr
43: require(!isAirdropMinted, 'already minted');

51: require(!isGenesisRewardMinted, 'already minted');

69: require(minterEnableDate <= block.timestamp, 'GolomToken: wa
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L88

```
File:   contracts/rewards/RewardDistributor.sol
88: require(msg.sender == trader);

144: require(epochs[index] < epoch);

158: require(epochs[index] < epoch);

173: require(address(ve) != address(0), ' VE not added yet');

181: require(tokenowner == ve.ownerOf(tokenids[tindex]), 'Can or

184: require(epochs[index] < epoch, 'cant claim for future epoch
185: require(claimed[tokenids[tindex]][epochs[index]] == 0, 'car

220: require(address(ve) != address(0), ' VE not added yet');

292: require(traderEnableDate <= block.timestamp, 'RewardDistrik

309: require(voteEscrowEnableDate <= block.timestamp, 'RewardDis
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/core/GolomTrader.sol#L177

```
File:   contracts/core/GolomTrader.sol
177: require(signaturesigner == o.signer, 'invalid signature');

211:  require(
212:    o.totalAmt >= o.exchange.paymentAmt + o.prePayment.payme
213:    'amt not matching'
214:  );
```

```
217: require(msg.value >= o.totalAmt * amount + p.paymentAmt, 'n

220: require(msg.sender == o.reservedAddress);

222: require(o.orderType == 0, 'invalid orderType');

226: require(status == 3, 'order not valid');
227: require(amountRemaining >= amount, 'order already filled');

235: require(amount == 1, 'only 1 erc721 at 1 time');

285: require(
286:   o.totalAmt * amount >
287:     (o.exchange.paymentAmt + o.prePayment.paymentAmt + o.ref

291:  require(msg.sender == o.reservedAddress);

293: require(o.orderType == 1);

295:  require(status == 3);
296:  require(amountRemaining >= amount);

299: require(amount == 1, 'only 1 erc721 at 1 time');

313: require(o.signer == msg.sender);

342: require(o.totalAmt >= o.exchange.paymentAmt + o.prePayment.

345: require(msg.sender == o.reservedAddress);

347: require(o.orderType == 2);

349: require(status == 3);
350: require(amountRemaining >= amount);

359: require(amount == 1, 'only 1 erc721 at 1 time');

455: require(distributorEnableDate <= block.timestamp, 'not allo
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowDelegation.sol#L72

```
File:   contracts/vote-escrow/VoteEscrowDelegation.sol
72: require(ownerOf(tokenId) == msg.sender, 'VEDelegation: Not a
73: require(this.balanceOfNFT(tokenId) >= MIN_VOTING_POWER_REQUI

99: require(_delegatedTokenIds.length < 500, 'VVDelegation: Cann

130: require(blockNumber < block.number, 'VEDelegation: not yet

186: require(blockNumber < block.number, 'VEDelegation: not yet

211: require(ownerOf(tokenId) == msg.sender, 'VEDelegation: Not

239: require(attachments[_tokenId] == 0 && !voted[_tokenId], 'at

245: require(_isApprovedOrOwner(_sender, _tokenId));
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L360

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
360: require(_entered_state == _not_entered);

538: require(attachments[_tokenId] == 0 && !voted[_tokenId], 'at

540: require(_isApprovedOrOwner(_sender, _tokenId));

646: require(owner != address(0));

648: require(_approved != owner);

652: require(senderIsOwner || senderIsApprovedForAll);

869: require(msg.sender == voter);

874: require(msg.sender == voter);

879: require(msg.sender == voter);

884: require(msg.sender == voter);

889: require(msg.sender == voter);
```

```
894: require(attachments[_from] == 0 && !voted[_from], 'attachec
895: require(_from != _to);
896: require(_isApprovedOrOwner(msg.sender, _from));
897: require(_isApprovedOrOwner(msg.sender, _to));

927: require(_value > 0); // dev: need non-zero value
928: require(_locked.amount > 0, 'No existing lock found');
929: require(_locked.end > block.timestamp, 'Cannot add to expir

944: require(_value > 0); // dev: need non-zero value
945: require(unlock_time > block.timestamp, 'Can only lock until
946: require(unlock_time <= block.timestamp + MAXTIME, 'Voting l

982: require(_locked.amount > 0, 'No existing lock found');
983: require(_locked.end > block.timestamp, 'Cannot add to expir

996: require(_locked.end > block.timestamp, 'Lock expired');
997: require(_locked.amount > 0, 'Nothing is locked');
998: require(unlock_time > _locked.end, 'Can only increase lock
999: require(unlock_time <= block.timestamp + MAXTIME, 'Voting l

1008: require(attachments[_tokenId] == 0 && !voted[_tokenId], 'a

1011: require(block.timestamp >= _locked.end, "The lock didn't e

1082: require(idToOwner[_tokenId] != address(0), 'Query for none

1227: require(_isApprovedOrOwner(msg.sender, _tokenId), 'caller
```

Mitigation

Custom errors are defined using the `error` statement

Replace `require` statements with custom errors.

[G-22] Multiple `address` mappings can be combined into a single `mapping` of an `address` to a `struct`, where appropriate

Saves a storage slot for the mapping. Depending on the circumstances and sizes of types, can avoid a Gsset (**20000 gas**) per mapping combined. Reads and subsequent writes can also be cheaper when a function requires both values and they both fit in the same storage slot. Finally, if both fields are accessed in the same function, can save **~42 gas per access** due to not having to recalculate the key's keccak256 hash (Gkeccak256 - **30 gas**) and that calculation's associated stack operations.

There are 5 instances of this issue:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L58

```
File:   contracts/rewards/RewardDistributor.sol
58:    mapping(address => mapping(uint256 => uint256)) public fe
59:    mapping(address => mapping(uint256 => uint256)) public fe
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L332

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
331: /// @dev Mapping from owner address to count of his tokens.
332: mapping(address => uint256) internal ownerToNFTokenCount;

334: /// @dev Mapping from owner address to mapping of index to
335: mapping(address => mapping(uint256 => uint256)) internal ow

340: /// @dev Mapping from owner address to mapping of operator
341: mapping(address => mapping(address => bool)) internal owner
```

[G-23] Functions guaranteed to revert when called by normal users can be marked `payable`

If a function modifier such as `onlyOwner` is used, the function will revert if a normal user tries to pay the function. Marking the function as `payable` will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided. The extra opcodes avoided are `CALLVALUE` (2), `DUP1` (3), `ISZERO` (3), `PUSH2` (3), `JUMPI` (10), `PUSH1` (3), `DUP1` (3), `REVER`

`T` (0), `JUMPDEST` (1), `POP` (2), which costs an average of about 21 gas per call to the function, in addition to the extra deployment cost.

There are 12 instances of this issue:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/governance/GolomToken.sol#L36

```
File:   contracts/governance/GolomToken.sol
36: function mint(address _account, uint256 _amount) external or

42: function mintAirdrop(address _airdrop) external onlyOwner {

50: function mintGenesisReward(address _rewardDistributor) exter

58: function setMinter(address _minter) external onlyOwner {

65: function executeSetMinter() external onlyOwner {
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L285

```
File:   contracts/rewards/RewardDistributor.sol
285: function changeTrader(address _trader) external onlyOwner {

291: function executeChangeTrader() external onlyOwner {

298: function addVoteEscrow(address _voteEscrow) external onlyOw

308: function executeAddVoteEscrow() external onlyOwner {
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/core/GolomTrader.sol#L444

```
File:   contracts/core/GolomTrader.sol
```

```
444: function setDistributor(address _distributor) external onl
454: function executeSetDistributor() external onlyOwner {
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowDelegation.sol#L260

```
File:   contracts/vote-escrow/VoteEscrowDelegation.sol
260: function changeMinVotingPower(uint256 _newMinVotingPower) e
```

[G-24] Do not calculate constants

Due to how constant variables are implemented (replacements at compile-time), an expression assigned to a constant variable is recomputed each time that the variable is used, which wastes some gas.
Instances include:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L48

```
File:   contracts/rewards/RewardDistributor.sol
48: uint256 constant dailyEmission = 600000 * 10**18;

57: uint256 constant secsInDay = 24 * 60 * 60;
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L296

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
296: uint256 internal constant MAXTIME = 4 * 365 * 86400;
297: int128 internal constant iMAXTIME = 4 * 365 * 86400;
```

[G-25] `<array>.length` should not be looked up in every loop of a for-loop

The overheads outlined below are *PER LOOP*, excluding the first loop

- storage arrays incur a Gwarmaccess (100 gas)
- memory arrays use `MLOAD` (3 gas)
- calldata arrays use `CALLDATALOAD` (3 gas)

Caching the length changes each of these to a `DUP<N>` (3 gas), and gets rid of the extra `DUP<N>` needed to store the stack offset.
Instances include:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L143

```
File:   contracts/rewards/RewardDistributor.sol
143: for (uint256 index = 0; index < epochs.length; index++) {

157: for (uint256 index = 0; index < epochs.length; index++) {

180: for (uint256 tindex = 0; tindex < tokenids.length; tindex++

183: for (uint256 index = 0; index < epochs.length; index++) {
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/core/GolomTrader.sol#L415

```
File:   contracts/core/GolomTrader.sol
415: for (uint256 i = 0; i < proof.length; i++) {
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowDelegation.sol#L171

```
File:   contracts/vote-escrow/VoteEscrowDelegation.sol
171: for (uint256 index = 0; index < delegated.length; index++)

189: for (uint256 index = 0; index < delegatednft.length; index+

199: for (uint256 i; i < _array.length; i++) {
```

[G-26] Copying struct to memory can be more expensive than just reading from storage

I suggest using the `storage` keyword instead of the `memory` one, as the copy in memory is wasting some MSTOREs and MLOADs.
Instances include:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L925

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
1083: LockedBalance memory _locked = locked[_tokenId];

1136: Point memory point_1 = point_history[_epoch + 1];
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowDelegation.sol#L101

```
File:   contracts/vote-escrow/VoteEscrowDelegation.sol
101: Checkpoint memory oldCheckpoint = checkpoints[toTokenId][n
```

Should use `storage`, because each struct field is read only once.

[G-27] `require()` / `revert()` strings longer than 32 bytes cost extra gas

Each extra chunk of byetes past the original 32 i[incurs an MSTORE](#) which costs 3 gas.
There are 8 instances of this issue:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/governance/GolomToken.sol#L24

```
File:   contracts/governance/GolomToken.sol
24: require(msg.sender == minter, 'GolomToken: only reward distr
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L181

```
File:   contracts/rewards/RewardDistributor.sol
181:  require(tokenowner == ve.ownerOf(tokenids[tindex]), 'Can c

292: require(traderEnableDate <= block.timestamp, 'RewardDistrib

309: require(voteEscrowEnableDate <= block.timestamp, 'RewardDis
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowDelegation.sol#L73

```
File:   contracts/vote-escrow/VoteEscrowDelegation.sol
73: require(this.balanceOfNFT(tokenId) >= MIN_VOTING_POWER_REQUI
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L929

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
929: require(_locked.end > block.timestamp, 'Cannot add to expir
945: require(unlock_time > block.timestamp, 'Can only lock unti

983: require(_locked.end > block.timestamp, 'Cannot add to expi
```

[G-28] Remove or replace unused variables

Remove or replace unused variables to save deployment gas.
Instances include:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L4

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
319:    string public constant version = '1.0.0';
320:    uint8 public constant decimals = 18;
```

[G-29] State variables only set in the constructor should be declared `immutable`

Avoids a Gsset (20000 gas) in the constructor, and replaces each Gwarmacces (100 gas) with a `PUSH32` (3 gas). If getters are still desired, '_' can be added to the variable name and the getter can be added manually.

There is 3 instance of this issue:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L143

```
File:   contracts/rewards/RewardDistributor.sol
68: ERC20 public rewardToken;
69: ERC20 public weth;
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-

escrow/VoteEscrowCore.sol#L300

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
300: address public token;
```

[G-30] State variables with values known at compile time should be constants

Variables with values known at compile time and that do not change at runtime should be declared as `constant`.
There is 2 instance of this issue:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L46

```
File:   contracts/rewards/RewardDistributor.sol
46: uint256 public startTime; // timestamp at which the contract
```

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/core/GolomTrader.sol#L45

```
File:   contracts/core/GolomTrader.sol
45: ERC20 WETH = ERC20(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc
```

[G-31] State variables should be cached in stack variables rather than re-reading them from storage

Caching will replace each Gwarmaccess (100 gas) with a much cheaper stack read. Less obvious fixes/optimizations include having local storage variables of mappings within state variable mappings or mappings within state variable structs, having local storage variables of structs within mappings, having local memory caches of state variable structs, or having local caches of state variable contracts/addresses.
Instances include:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L98-L138 `rewardToken` accessed 7 times; `epoch` accessed 15 times; `ve` accessed 2 times; `epochTotalFee[epoch]` accessed 2 times;

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L144 `epoch` accessed N times, each loop iteration, where N = epochs.length

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L158 `epoch` accessed N times, each loop iteration, where N = epochs.length

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L172-L210 `ve` accessed 8 times `epochBeginTime[epochs[index]]` accessed 4 times `epoch` accessed N times, each loop iteration, where N = epochs.length

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/core/GolomTrader.sol#L242-L269 `distributor` accessed two times;

Small PoC

copy `distributor` to stack variable:

`Distributor _distributor = distributor ;`

use `_distributor` instead of `distributor`;

`hardhat-gas-reporter` results

before fix: | Methods ||||| |-|:-|:-:|-|:-|:-:| | Contract | Method | min | max | avg | #calls | | GolomTrader | fillAsk | 238153 |241948|241401| 7 | after fix: | Methods ||||| |-|:-|:-:|-|:-|:-:| | Contract | Method | min | max | avg | #calls | | GolomTrader | fillAsk | 238052 |241847|241300| 7 |

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/core/Gol

omTrader.sol#L382-L383
`WETH` accessed two times;

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/core/GolomTrader.sol#L384-L402
`distributor` accessed two times;

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowDelegation.sol#L138-L139
`checkpoints[nftId][nCheckpoints - 1]` accessed two times;

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowDelegation.sol#L213-L214
`checkpoint.delegatedTokenIds` accessed two times;

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L644-L650
`idToOwner[_tokenId]` accessed two times;

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L948-L949
`tokenId` accessed two times;

[G-32] Using `private` rather than `public` for constants, saves gas

If needed, the value can be read from the verified contract source code. Savings are due to the compiler not having to create non-payable getter functions for deployment calldata, and not adding another entry to the method ID table.
Instances include:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-

escrow/VoteEscrowCore.sol#L317

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
317:     string public constant name = 'veNFT';
318:    string public constant symbol = 'veNFT';
319:    string public constant version = '1.0.0';
320:    uint8 public constant decimals = 18;
```

[G-33] Remove unreachable code

There is a peace of code that does not need to exist:
https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/core/GolomTrader.sol#L177

```
File:   contracts/core/GolomTrader.sol
177:        require(signaturesigner == o.signer, 'invalid signat
178:        if (signaturesigner != o.signer) {
179:            return (0, hashStruct, 0); //@audit unreachable
180:        }
```

Check inside `if` is pointless, because `signaturesigner != o.signer` can not be `true`, when if it is, then transaction is already reverted because of line 177.
`hardhat-gas-reporter` shows deployment gas difference from `2013842` to `2001108`.

[G-34] No need to evaluate all expressions to know if one of them is true

When we have a code `expressionA || expressionB` if `expressionA` is true then `expressionB` will not be evaluated and gas saved;
Instances include:

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L650

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
650: bool senderIsOwner = (idToOwner[_tokenId] == msg.sender);
651: bool senderIsApprovedForAll = (ownerToOperators[owner])[msg
652: require(senderIsOwner || senderIsApprovedForAll);
```

Variables `bool senderIsOwner` and `bool senderIsApprovedForAll` just add more work, it should be:

```
require((idToOwner[_tokenId] == msg.sender) || (ownerToOperator
```

so if `(idToOwner[_tokenId] == msg.sender)` is `true` we will not do SLOAD to get `(ownerToOperators[owner])[msg.sender]` ; saving runtime and deployment gas;

same thing here: https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L439

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
439:        bool spenderIsOwner = owner == _spender;
440:        bool spenderIsApproved = _spender == idToApprovals[_
441:        bool spenderIsApprovedForAll = (ownerToOperators[own
442:        return spenderIsOwner || spenderIsApproved || spende
```

But this one is a `view` function, so only deployment gas saved.

[G-35] No need to read `tokenId` second time

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L948

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
948:        ++tokenId;
949:        uint256 _tokenId = tokenId;
```

Mitigation

Change it to `uint256 _tokenId = ++tokenId;` and `92 gas is saved this way`

[G-36] `last_point` value rewritten right after initialization

If `_epoch > 0`, then `last_point` is rewritten, and initialization on L726 becomes waste of gas;
https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#726

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
726:        Point memory last_point = Point({bias: 0, slope: 0,
727:        if (_epoch > 0) {
728:            last_point = point_history[_epoch];
729:        }
```

Mitigation

```
Point memory last_point = _epoch > 0 ? last_point = point_histor
```

[G-37] Wasted gas on copying a struct

https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#840

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
840: LockedBalance memory _locked = locked_balance;

1164:     function _supply_at(Point memory point, uint256 t) int
1165:         Point memory last_point = point;
```

`locked_balance` is of same type and never used after this. Same thing with `point`; There is no need in copy of it, but gas for making a copy is spent;

[G-38] Remove duplicate code

There is a peace of code that is duplicated in bo blocks of `if else`
https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#467

```
467:    if (current_count == current_index) {
            // update ownerToNFTokenIdList
            ownerToNFTokenIdList[_from][current_count] = 0;
            // update tokenToOwnerIndex
            tokenToOwnerIndex[_tokenId] = 0;
        } else {
            uint256 lastTokenId = ownerToNFTokenIdList[_from][cu

            // Add
            // update ownerToNFTokenIdList
            ownerToNFTokenIdList[_from][current_index] = lastTok
            // update tokenToOwnerIndex
            tokenToOwnerIndex[lastTokenId] = current_index;

            // Delete
            // update ownerToNFTokenIdList
            ownerToNFTokenIdList[_from][current_count] = 0;
            // update tokenToOwnerIndex
            tokenToOwnerIndex[_tokenId] = 0;
        }
```

To reduce deployment gas should be:

```
        if (current_count != current_index) {
                        uint256 lastTokenId = ownerToNFTokenIdLi

                        // Add
                        // update ownerToNFTokenIdList
                        ownerToNFTokenIdList[_from][current_inde
                        // update tokenToOwnerIndex
                        tokenToOwnerIndex[lastTokenId] = current
```

```
            }
            // update ownerToNFTokenIdList
            ownerToNFTokenIdList[_from][current_count] = 0;
            // update tokenToOwnerIndex
            tokenToOwnerIndex[_tokenId] = 0;
```

[G-39] No need for mapping `supportedInterfaces` to exist

This mapping takes some space and every time `supportsInterface(bytes4)` is called we do `SLOAD` to get data from this mapping.
https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L368

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
368:     function supportsInterface(bytes4 _interfaceID) externa
369:         return supportedInterfaces[_interfaceID];
370:     }
```

I suggest we remove the mapping and change the function `supportsInterface(bytes4)` to:

```
368:     function supportsInterface(bytes4 _interfaceID) externa
369:         return _interfaceID == ERC165_INTERFACE_ID ||


370:     }
```

This way we don't need the mapping `supportedInterfaces` and no `SLOAD`s done; If inherited contracts support other interfaces or don't support these just override this function and add/remove conditions;

[G-40] Same calculation twice

`supply_before - value` calculated twice, also it is *checked* arithmetic operation, so cheaper to store result in a stack variable than calculate it twice;
https://github.com/code-423n4/2022-07-

golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L1015-L1029

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
1015:         uint256 supply_before = supply;
1016:         supply = supply_before - value;

1029:         emit Supply(supply_before, supply_before - value);
```

We could store result to stack variable and use it instead:

```
1016:         uint256 supply_now = supply = supply_before - valu

1029:         emit Supply(supply_before, supply_now);
```

[G-41] Obsolete constants

These values can be accessed via kewords, there is no benefit in having these constants:
https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/rewards/RewardDistributor.sol#L57

```
File:   contracts/rewards/RewardDistributor.sol
57:     uint256 constant secsInDay = 24 * 60 * 60;
```

We can use `1 days` instead of `secsInDay`
https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L4

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
295:     uint256 internal constant WEEK = 1 weeks;
```

We can use `1 weeks` instead of `WEEK`.

[G-42] Variable `end` can be of type `uint128`

Variable `end` can be of type `uint128` to take less place and save one storage slot. A lot of protocols even use `uint64` for dates to reduce gas usage.
https://github.com/code-423n4/2022-07-golom/blob/7bbb55fca61e6bae29e57133c1e45806cbb17aa4/contracts/vote-escrow/VoteEscrowCore.sol#L270-L273

```
File:   contracts/vote-escrow/VoteEscrowCore.sol
270: struct LockedBalance {
271:     int128 amount;
272:     uint256 end; //@audit can be uint128 to save storage sl
273: }
```

it can be

```
270: struct LockedBalance {
271:     int128 amount;
272:     uint128 end;
273: }
```

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top