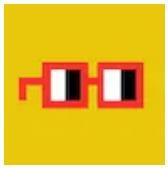

Nouns DAO contest
Findings & Analysis Report

2022-09-30

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 DAO smart contract system written in Solidity. The audit contest took place between August 22—August 27 2022.

Wardens

168 Wardens contributed reports to the Nouns DAO contest:

1. rbserver
2. Respx
3. Lambda
4. KIntern_NA (TrungOre and duc)
5. berndartmueller
6. csanuragjain
7. cccz
8. zzzitron
9. bin2chen

10. IEatBabyCarrots
11. jayphbee
12. Deivitto
13. 0xSmartContract
14. 0xDjango
15. Aymen0909
16. Ch_301
17. TomJ
18. lllllll
19. 0xNazgul
20. JohnSmith
21. mics
22. CodingNameKiki
23. Dravee
24. JC
25. 0x1f8b
26. Rolezn
27. m_Rassska
28. cRat1st0s
29. Bnke0x0
30. _141345_
31. GalloDaSballo
32. gogo
33. 0xNineDec
34. Funen
35. oyc_109
36. robee
37. ReyAdmirado
38. pfapostol

39. ElKu
40. c3phas
41. ajtra
42. Sm4rty
43. erictee
44. 0xkatana
45. sikorico
46. Olivierdem
47. carlitox477
48. saian
49. hyh
50. brgltd
51. bobirichman
52. seyni
53. ladboy233
54. fatherOfBlocks
55. prasantgupta52
56. ret2basic
57. Tomo
58. durianSausage
59. LeoS
60. sryysryy
61. simon135
62. GimelSec (rayn and sces60107)
63. catchup
64. Waze
65. delfin454000
66. Guardian
67. d3e4

68. lukris02
69. 0xbepresent
70. CertoraInc (egjlmn1, OriDabush, ItayG, shakedwinder, and RoiEvenHaim)
71. pauliax
72. Rohan16
73. rvierdiiev
74. 0x040
75. rfa
76. DimitarDimitrov
77. Ruhum
78. sach1r0
79. djxploit
80. RaymondFam
81. _Adam
82. Chom
83. SooYa
84. Bjorn_bug
85. R2
86. tnevler
87. mrpathfindr
88. natzuu
89. DevABDee
90. Saintcode_
91. rokinot
92. Noah3o6
93. wagmi
94. auditor0517
95. Jeiwan
96. xiaoming90

97. 0bi
98. 0x1337
99. rajatbeladiya
100. sseefried
101. exd0tpy
102. 0xRajeev
103. dipp
104. 0xSky
105. asutorufos
106. Soosh
107. yixxas
108. tonisives
109. shenwilly
110. p_crypt0
111. zkhorse (karmacoma and horsefacts)
112. JansenC
113. 0xmatt
114. pashov
115. Haruxe
116. android69
117. 8olidity
118. Trabajo_de_mates (Saintcode_ and tay054)
119. z3s
120. throttle
121. joestakey
122. martin
123. Junnon
124. ch0bu
125. samruna

126. jag
127. Shishigami
128. Ben
129. ignacio
130. SaharAP
131. BipinSah
132. bulej93
133. lucacez
134. exolorkistis
135. zishansami
136. Tomio
137. Fitraldys
138. medikko
139. EthLedger
140. shr1ftyy
141. rotcivegaf
142. ak1
143. karanctf
144. shark
145. 0xc0ffEE
146. Amithuddar
147. SerMyVillage
148. 2997ms
149. newfork01
150. RoiEvenHaim
151. Polandia94
152. tay054
153. Yiko
154. ACai

155. francoHacker
156. Randyyy
157. Diraco
158. IgnacioB
159. peritoflores
160. a12jmx

This contest was judged by gzeon.

Final report assembled by itsmetechjay.

Summary

The C4 analysis yielded an aggregated total of 4 unique vulnerabilities. Of these vulnerabilities, 1 received a risk rating in the category of HIGH severity and 3 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 116 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 126 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 DAO contest repository, and is composed of 6 smart contracts written in the Solidity programming language and includes 2,412 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 (1)

[H-01] ERC721Checkpointable: delegateBySig allows the user to vote to address 0, which causes the user to permanently lose his vote and cannot transfer his NFT.

Submitted by cccz, also found by berndartmueller, bin2chen, csanuragjain, IEatBabyCarrots, jayphbee, KIntern_NA, Lambda, and zzzitron

In the ERC721Checkpointable contract, when the user votes with the delegate function, the delegatee will not be address 0.

```
function delegate(address delegatee) public {
    if (delegatee == address(0)) delegatee = msg.sender;
    return _delegate(msg.sender, delegatee);
}
```

However, there is no such restriction in the delegateBySig function, which allows the user to vote to address 0.

```
function delegateBySig(
    address delegatee,
    uint256 nonce,
    uint256 expiry,
```

```
        uint8 v,
        bytes32 r,
        bytes32 s
    ) public {
        bytes32 domainSeparator = keccak256(
            abi.encode(DOMAIN_TYPEHASH, keccak256(bytes(name())))
        );
        bytes32 structHash = keccak256(abi.encode(DELEGATION_TYP
        bytes32 digest = keccak256(abi.encodePacked('\x19\x01',
        address signatory = ecrecover(digest, v, r, s);
        require(signatory != address(0), 'ERC721Checkpointable::
        require(nonce == nonces[signatory]++, 'ERC721Checkpointa
        require(block.timestamp <= expiry, 'ERC721Checkpointable
        return _delegate(signatory, delegatee);
    }
```

If user A votes to address 0 in the delegateBySig function, _delegates[A] will be address 0, but the delegates function will return the address of user A and getCurrentVotes(A) will return 0.

```
    function _delegate(address delegator, address delegatee) int
        /// @notice differs from `_delegate()` in `Comp.sol` to
        address currentDelegate = delegates(delegator);

        _delegates[delegator] = delegatee;
...
    function delegates(address delegator) public view returns (a
        address current = _delegates[delegator];
        return current == address(0) ? delegator : current;
    }
```

Later, if user A votes to another address or transfers NFT, the _moveDelegates function will fail due to overflow, which makes user A lose votes forever and cannot transfer NFT.

```
    function _moveDelegates(
        address srcRep,
        address dstRep,
        uint96 amount
    ) internal {
        if (srcRep != dstRep && amount > 0) {
```

```
            if (srcRep != address(0)) {
                uint32 srcRepNum = numCheckpoints[srcRep];
                uint96 srcRepOld = srcRepNum > 0 ? checkpoints[s
                uint96 srcRepNew = sub96(srcRepOld, amount, 'ERC
                _writeCheckpoint(srcRep, srcRepNum, srcRepOld, s
            }
```

On the other hand, since the burn function also fails, this can also be used to prevent the NFT from being burned by the minter

```
    function burn(uint256 nounId) public override onlyMinter {
        _burn(nounId);
        emit NounBurned(nounId);
    }
...
    function _burn(uint256 tokenId) internal virtual {
        address owner = ERC721.ownerOf(tokenId);

        _beforeTokenTransfer(owner, address(0), tokenId);
...
    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 tokenId
    ) internal override {
        super._beforeTokenTransfer(from, to, tokenId);

        /// @notice Differs from `_transferTokens()` to use `del
        _moveDelegates(delegates(from), delegates(to), 1);
    }
```

Proof of Concept

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/base/ERC721Checkpointable.sol#L126-L144

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/base/ERC721Checkpointable.sol#L88-L91

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/base/ERC721Checkpointable.sol#L97-L106

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/base/ERC721Checkpointable.sol#L197-L208

Recommended Mitigation Steps

Consider requiring in the `delegateBySig` function that delegatee cannot be address 0.

```
        function delegateBySig(
            address delegatee,
            uint256 nonce,
            uint256 expiry,
            uint8 v,
            bytes32 r,
            bytes32 s
        ) public {
    + require(delegatee != address(0));
```

eladmallel (Nouns DAO) confirmed and commented:

> We agree this is a bug that has existed since Nouns launched, and plan to fix the bug with the suggested requirement that delegatee should not be address(0).

> Worth noting that this fix will not have a positive effect on Nouns, as the token is already deployed and not upgradable.

Medium Risk Findings (3)

[M-01] Voters can burn large amounts of Ether by submitting votes with long reason strings

Submitted by Respx

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L518-L524

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L533-L544

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L98

Vulnerability Details

Voters can burn large amounts of Ether by submitting votes with long reason strings

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L518-L524

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L533-L544

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L98

The only limits to how long a string argument to a function call can be is the block gas limit of the EVM, currently 30 million. It is therefore possible for a user to call `NounsDAOLogicV2.castRefundableVoteWithReason()` with a very, very long `reason` string. `castRefundableVoteInternal()` emits an event that includes `reason` on line 540, which is within the region of code covered by gas refunds (gas refunds are measured from `startGas` on line 538). Because of this, gas refunds will include the gas price of emitting this event, which could potentially be very large.

Impact

This issue is partially mitigated by the fact that the voter will still bear the cost of the massive calldata usage. `NounsDAOLogicV2` covers this with a fixed value of `REFUND_BASE_GAS` (36000), but the real transaction overhead is far larger when submitting a `reason` string that is many thousands of characters in length. Therefore, the voter ends up losing roughly as much as is drained from the `NounsDAOLogicV2` contract by the refund. Nonetheless, I still think this is a valid high funding as the protocol will not want to rely purely on this economic protection. Some risk scenarios:

1. It is quite possible that calldata prices could decrease in future, perhaps as part of catering for rollups. This could make the attack suddenly far more viable.
2. A voter might have some motive to want to emit some arbitrary text as an Ethereum event, and simply exploit this system to do so.
3. A voter might want to maliciously drain the Ether, perhaps to damage the protocol's reputation.
4. As in 3, this could be achieved by emptying out the last funds in ŃounsDAOLogicV2`and so denying many other voters their voting refunds.

Tools Used

Hardhat testing

Recommended Mitigation Steps

2 alternative ideas:

1. Move the `emit VoteCast` outside of the gas calculation region of the code and increase `REFUND_BASE_GAS` to cover an event with a reasonable length of string.
2. Change the type of `reason` to `bytes` and add a check to its length in `castRefundableVoteWithReason()` , reverting if it is too long.

Proof of Concept

The single vote in this test burns around 0.25 Ether from the `NounsDAOLogicV2` contract. This test runs slowly and is assuming a base fee of 500 gwei. Obviously if the base fee were higher, the gas burnt would also be higher. The numbers are printed out with a rather messy `console.log()` in the middle of the test output.

Apologies for the bad presentation, but on the bright side you can adjust the numbers and see different results.

```
diff --git a/hardhat.config.ts b/hardhat.config.ts
index 6d469b0..dc51148 100644
--- a/hardhat.config.ts
+++ b/hardhat.config.ts
@@ -34,7 +34,7 @@ const config: HardhatUserConfig = {
         : [process.env.WALLET_PRIVATE_KEY!].filter(Boolean),
     },
     hardhat: {
-      initialBaseFeePerGas: 0,
+      initialBaseFeePerGas: 500_000_000_000,
     },
   },
   etherscan: {
@@ -50,12 +50,12 @@ const config: HardhatUserConfig = {
   gasReporter: {
     enabled: !process.env.CI,
     currency: 'USD',
-    gasPrice: 50,
+    gasPrice: 500,
     src: 'contracts',
     coinmarketcap: '7643dfc7-a58f-46af-8314-2db32bdd18ba',
   },
   mocha: {
-    timeout: 60_000,
+    timeout: 600_000,
   },
 };
 export default config;
```

```
diff --git a/test/governance/NounsDAO/V2/voteRefund.test.ts b/te
index d34ff4b..4c268a3 100644
--- a/test/governance/NounsDAO/V2/voteRefund.test.ts
+++ b/test/governance/NounsDAO/V2/voteRefundPOC.test.ts
@@ -162,6 +162,30 @@ describe('Vote Refund', () => {
   });

   describe('castRefundableVoteWithReason', () => {
+    it("accepts excessively long reason strings", async () => {
+      await fundGov();
+      const balanceBefore = await user.getBalance();
```

```
+      const govBalanceBefore = await ethers.provider.getBalance
+      const tx = await gov
+        .connect(user)
+        .castRefundableVoteWithReason(1, 1, junkString(50_000),
+          maxPriorityFeePerGas: MAX_PRIORITY_FEE_CAP,
+          gasLimit: 24000000,
+        });
+      const r = await tx.wait();
+      const balanceDiff = balanceBefore.sub(await user.getBalar
+      const govBalanceDiff = govBalanceBefore.sub(
+        await ethers.provider.getBalance(gov.address)
+      );
+      const govBalanceAfter = await ethers.provider.getBalance
+      console.log("USER BALANCE DIFF:", ethers.utils.formatEthe
+      console.log(
+        "GOV BALANCE DIFF:",
+        ethers.utils.formatEther(govBalanceDiff)
+      );
+      console.log("TX COST:", ethers.utils.formatEther(await tx
+
+    });
     it('refunds users with votes', async () => {
       await fundGov();
       const balanceBefore = await user.getBalance();
@@ -284,6 +308,15 @@ describe('Vote Refund', () => {
     expect(refundEvent!.args!.refundAmount).to.be.closeTo(expec
   }

+  function junkString(iterations: number = 100) {
+    var x = "Ab Cd Ef Gh Ij ";
+    const y = "Ab Cd Ef Gh Ij";
+    for (var i = 0; i < iterations; i++) {
+      x += y;
+    }
+    return x;
+  }
+
   async function submitProposal(u: SignerWithAddress) {
     await gov
       .connect(u)
```

[eladmallel (Nouns DAO) disagreed with severity and commented]():

> We acknowledge that a Noun holder can push the refund amount up with a long reason string. We think this is low risk since again this is capped by the number of proposals one can vote on, furthermore buying an expensive Noun just to perform this no-profit attack seems unlikely at the moment.

> Having said that, we do plan to mitigate this concern by adding a cap on the `gasUsed` variable used in the refund calculation.

gzeoneth (judge) decreased severity to Medium

[M-02] User A cannot cancel User B's proposal when User B's prior number of votes at relevant block is same as proposal threshold, which contradicts the fact that User B actually cannot create the proposal when the prior number of votes is same as proposal threshold

Submitted by rbserver

https://github.com/code-423n4/2022-08-nounsdao/blob/main/contracts/governance/NounsDAOLogicV2.sol#L184-L279

https://github.com/code-423n4/2022-08-nounsdao/blob/main/contracts/governance/NounsDAOLogicV2.sol#L346-L368

Impact

When User B calls the following `propose` function for creating a proposal, it checks that User B's prior number of votes at the relevant block is larger than the proposal threshold through executing `nouns.getPriorVotes(msg.sender, block.number - 1) > temp.proposalThreshold`. This means that User B cannot create the proposal when the prior number of votes and the proposal threshold are the same.

https://github.com/code-423n4/2022-08-nounsdao/blob/main/contracts/governance/NounsDAOLogicV2.sol#L184-L279

```
    function propose(
        address[] memory targets,
```

```
        uint256[] memory values,
        string[] memory signatures,
        bytes[] memory calldatas,
        string memory description
    ) public returns (uint256) {
        ProposalTemp memory temp;

        temp.totalSupply = nouns.totalSupply();

        temp.proposalThreshold = bps2Uint(proposalThresholdBPS,

        require(
            nouns.getPriorVotes(msg.sender, block.number - 1) >
            'NounsDAO::propose: proposer votes below proposal th
        );
        require(
            targets.length == values.length &&
                targets.length == signatures.length &&
                targets.length == calldatas.length,
            'NounsDAO::propose: proposal function information ar
        );
        require(targets.length != 0, 'NounsDAO::propose: must pr
        require(targets.length <= proposalMaxOperations, 'NounsD

        temp.latestProposalId = latestProposalIds[msg.sender];
        if (temp.latestProposalId != 0) {
            ProposalState proposersLatestProposalState = state(t
            require(
                proposersLatestProposalState != ProposalState.Ac
                'NounsDAO::propose: one live proposal per propos
            );
            require(
                proposersLatestProposalState != ProposalState.Pe
                'NounsDAO::propose: one live proposal per propos
            );
        }

        temp.startBlock = block.number + votingDelay;
        temp.endBlock = temp.startBlock + votingPeriod;

        proposalCount++;
        Proposal storage newProposal = _proposals[proposalCount]
        newProposal.id = proposalCount;
        newProposal.proposer = msg.sender;
        newProposal.proposalThreshold = temp.proposalThreshold;
        newProposal.eta = 0;
```

```
        newProposal.targets = targets;
        newProposal.values = values;
        newProposal.signatures = signatures;
        newProposal.calldatas = calldatas;
        newProposal.startBlock = temp.startBlock;
        newProposal.endBlock = temp.endBlock;
        newProposal.forVotes = 0;
        newProposal.againstVotes = 0;
        newProposal.abstainVotes = 0;
        newProposal.canceled = false;
        newProposal.executed = false;
        newProposal.vetoed = false;
        newProposal.totalSupply = temp.totalSupply;
        newProposal.creationBlock = block.number;

        latestProposalIds[newProposal.proposer] = newProposal.ic

        /// @notice Maintains backwards compatibility with Gover
        emit ProposalCreated(
            newProposal.id,
            msg.sender,
            targets,
            values,
            signatures,
            calldatas,
            newProposal.startBlock,
            newProposal.endBlock,
            description
        );

        /// @notice Updated event with `proposalThreshold` and `
        /// @notice `minQuorumVotes` is always zero since V2 int
        emit ProposalCreatedWithRequirements(
            newProposal.id,
            msg.sender,
            targets,
            values,
            signatures,
            calldatas,
            newProposal.startBlock,
            newProposal.endBlock,
            newProposal.proposalThreshold,
            minQuorumVotes(),
            description
        );
```

```
        return newProposal.id;
    }
```

After User B's proposal is created, User A can call the following `cancel` function to cancel it. When calling `cancel`, it checks that User B's prior number of votes at the relevant block is less than the proposal threshold through executing `nouns.getPriorVotes(proposal.proposer, block.number - 1) < proposal.proposalThreshold`. When User B's prior number of votes and the proposal threshold are the same, User A cannot cancel this proposal of User B. However, this contradicts the fact User B actually cannot create this proposal when the same condition holds true. In other words, if User B cannot create this proposal when the prior number of votes and the proposal threshold are the same, User A should be able to cancel User B's proposal under the same condition but it is not true. The functionality for canceling User B's proposal in this situation becomes unavailable for User A.

https://github.com/code-423n4/2022-08-nounsdao/blob/main/contracts/governance/NounsDAOLogicV2.sol#L346-L368

```
    function cancel(uint256 proposalId) external {
        require(state(proposalId) != ProposalState.Executed, 'Nc

        Proposal storage proposal = _proposals[proposalId];
        require(
            msg.sender == proposal.proposer ||
                nouns.getPriorVotes(proposal.proposer, block.num
            'NounsDAO::cancel: proposer above threshold'
        );

        proposal.canceled = true;
        for (uint256 i = 0; i < proposal.targets.length; i++) {
            timelock.cancelTransaction(
                proposal.targets[i],
                proposal.values[i],
                proposal.signatures[i],
                proposal.calldatas[i],
                proposal.eta
            );
        }

        emit ProposalCanceled(proposalId);
```

```
}
```

Proof of Concept

Please append the following test in the `NounsDAOV2#inflationHandling describe` block in `test\governance\NounsDAO\V2\inflationHandling.test.ts`. This test should pass to demonstrate the described scenario.

```
it("User A cannot cancel User B's proposal when User B's prior
  async () => {
  // account1 has 3 tokens at the beginning
  // account1 gains 2 more to own 5 tokens in total
  await token.transferFrom(deployer.address, account1.address,
  await token.transferFrom(deployer.address, account1.address,

  await mineBlock();

  // account1 cannot create a proposal when owning 5 tokens ir
  await expect(
    gov.connect(account1).propose(targets, values, signatures,
  ).to.be.revertedWith('NounsDAO::propose: proposer votes belc

  // account1 gains 1 more to own 6 tokens in total
  await token.transferFrom(deployer.address, account1.address,

  await mineBlock();

  // account1 can create a proposal when owning 6 tokens in tc
  await gov.connect(account1).propose(targets, values, signatu
  const proposalId = await gov.latestProposalIds(account1.addr
  expect(await gov.state(proposalId)).to.equal(0);

  // other user cannot cancel account1's proposal at this mome
  await expect(
    gov.cancel(proposalId, {gasLimit: 1e6})
  ).to.be.revertedWith('NounsDAO::cancel: proposer above thres

  // account1 removes 1 token to own 5 tokens in total
  await token.connect(account1).transferFrom(account1.address,

  await mineBlock();
```

```
    // other user still cannot cancel account1's proposal when a
    // this contradicts the fact that account1 cannot create a p
    await expect(
      gov.cancel(proposalId, {gasLimit: 1e6})
    ).to.be.revertedWith('NounsDAO::cancel: proposer above thres

    // account1 removes another token to own 4 tokens in total
    await token.connect(account1).transferFrom(account1.address,

    await mineBlock();

    // other user can now cancel account1's proposal when accour
    await gov.cancel(proposalId, {gasLimit: 1e6})
    expect(await gov.state(proposalId)).to.equal(2);
  });
```

Tools Used

VSCode

Recommended Mitigation Steps

https://github.com/code-423n4/2022-08-nounsdao/blob/main/contracts/governance/NounsDAOLogicV2.sol#L197-L200 can be changed to the following code.

```
        require(
            nouns.getPriorVotes(msg.sender, block.number - 1) >=
            'NounsDAO::propose: proposer votes below proposal th
        );
```

or

https://github.com/code-423n4/2022-08-nounsdao/blob/main/contracts/governance/NounsDAOLogicV2.sol#L350-L354 can be changed to the following code.

```
        require(
            msg.sender == proposal.proposer ||
                nouns.getPriorVotes(proposal.proposer, block.num
```

```
            'NounsDAO::cancel: proposer above threshold'
        );
```

but not both.

[eladmallel (Nouns DAO) confirmed and commented]():

> We agree that the case of prior votes equal to `proposalThreshold` is missed, and plan to include that state in what is cancelable.

[M-03] Loss of Veto Power can Lead to 51% Attack

Submitted by TomJ, also found by 0xDjango, 0xSmartContract, Aymen0909, Ch_301, and Deivitto

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L156

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV1.sol#L150

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L839-L845

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV1.sol#L637-L643

Impact

The veto power is import functionality for current NounsDAO 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 Nounders losing their veto power unintentionally and open to 51% attack which can drain their entire treasury.

Refrence from Nouns DAO contest documents: 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()` and `\_setVetoer()` of NounsDAOLogicV2.sol and NounsDAOLogicV1.sol. Also it is better to make changing address process of vetoer at `\_setVetoer()` into 2-step process to avoid accidently setting vetoer to zero address or any other arbitrary addresses and end up burning/losing veto power unintentionally.

1. Vetoer address of `initialize()` of NounsDAOLogicV2.sol, NounsDAOLogicV1.sol

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L156

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV1.sol#L150

2. Vetoer address of `\_setVetoer()` of NounsDAOLogicV2.sol, NounsDAOLogicV1.sol

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L839-L845

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV1.sol#L637-L643

Recommended Mitigation Steps

Add zero address check for vetoer address at `initialize()`. Also change `\_setVetoer()` vetoer address changing process to 2-step process like explained below.

First make the `\_setVetoer()` 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.

eladmallel (Nouns DAO) confirmed and commented:

> We agree it's worth being extra safe here, planning to change it to a 2-step process.

Low Risk and Non-Critical Issues

For this contest, 116 reports were submitted by wardens detailing low risk and non-critical issues. The **report highlighted below** by **IIIIIII** received the top score from the judge.

The following wardens also submitted reports: 0xNazgul, Deivitto, mics, CodingNameKiki, JC, 0xSmartContract, Lambda, Rolezn, rbserver, Dravee, Bnke0x0, 0x1f8b, _141345_, 0xNineDec, 0xDjango, Ch_301, auditor0517, Funen, GalloDaSballo, gogo, oyc_109, carlitox477, bobirichman, sikorico, ElKu, seyni, robee, saian, Aymen0909, c3phas, Olivierdem, hyh, brgltd, durianSausage, LeoS, Jeiwan, ladboy233, xiaoming90, simon135, sryysryy, GimelSec, catchup, cccz, Waze, berndartmueller, ajtra, delfin454000, Guardian, d3e4, lukris02, csanuragjain, 0bi, ReyAdmirado, fatherOfBlocks, 0x1337, djxploit, Bjorn_bug, pfapostol, 0xbepresent, RaymondFam, rajatbeladiya, zzzitron, cRat1st0s, CertoraInc, _Adam, sseefried, Sm4rty, exd0tpy, KIntern_NA, 0xRajeev, Chom, JohnSmith, dipp, pauliax, R2, Rohan16, 0xSky, TomJ, SooYa, tnevler, asutorufos, mrpathfindr, Soosh, yixxas, rvierdiiev, tonisives, 0x040, prasantgupta52, ret2basic, shenwilly, p_crypt0, natzuu, zkhorse, JansenC, 0xmatt, rfa, wagmi, pashov, erictee, DimitarDimitrov, DevABDee, Haruxe, Saintcode_, android69, rokinot, 8olidity, Trabajo_de_mates, Ruhum, z3s, 0xkatana, throttle, sach1r0, Noah3o6, Respx, *and* Tomo.

Low Risk Issues

	Issue	Instances
[L-01]	Nouns will not be able to be transferred once the `block.number` passes `type(uint32).max`	2
[L-02]	Unused/empty `receive()` / `fallback()` function	1
[L-03]	Missing checks for `address(0x0)` when assigning values to `address` state variables	5

Total: 8 instances over 3 issues

[L-01] Nouns will not be able to be transferred once the `block.number` passes `type(uint32).max`

While this currently equates to ~1260 years, if there's a hard fork which makes block times much more frequent (e.g. to compete with Solana), then this limit may be reached much faster than expected, and transfers and delegations will remain stuck at their existing settings

There are 2 instances of this issue:

```
File: /contracts/base/ERC721Checkpointable.sol

238           uint32 blockNumber = safe32(
239               block.number,
240               'ERC721Checkpointable::_writeCheckpoint: block
241:          );
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/base/ERC721Checkpointable.sol#L238-L241

```
File: /contracts/governance/NounsDAOLogicV2.sol

923:          uint32 blockNumber = safe32(blockNumber_, 'NounsDAO
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L923

[L-02] Unused/empty `receive()` / `fallback()` function

If the intention is for the Ether to be used, the function should call another function, otherwise it should revert (e.g. `require(msg.sender == address(weth))`). Having no access control on the function means that someone may send Ether to the contract, and have no way to get anything back out, which is a loss of funds

There is 1 instance of this issue:

```
File: contracts/governance/NounsDAOLogicV2.sol

1030:     receive() external payable {}
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L1030

[L-03] Missing checks for `address(0x0)` when assigning values to `address` state variables

There are 5 instances of this issue:

```
File: contracts/governance/NounsDAOLogicV1.sol

605:           pendingAdmin = newPendingAdmin;

642:           vetoer = newVetoer;
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV1.sol#L605

```
File: contracts/governance/NounsDAOLogicV2.sol

807:           pendingAdmin = newPendingAdmin;

844:           vetoer = newVetoer;
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L807

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

71:           admin = admin_;
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOProxy.sol#L71

Non-Critical Issues

	Issue	Instances
[N-01]	`public` functions not called by the contract should be declared `external` instead	8
[N-02]	Non-assembly method available	3
[N-03]	`2**<n> - 1` should be re-written as `type(uint<n>).max`	2
[N-04]	`constant` s should be defined rather than using magic numbers	8
[N-05]	Use a more recent version of solidity	3
[N-06]	Expressions for constant values such as a call to `keccak256()`, should use `immutable` rather than `constant`	6
[N-07]	Constant redefined elsewhere	11

	Issue	Instances
[N-08]	Lines are too long	7
[N-09]	Non-library/interface files should use fixed compiler versions, not floating ones	4
[N-10]	Event is missing `indexed` fields	21
[N-11]	Not using the named return variables anywhere in the function is confusing	8
[N-12]	Typos	4

Total: 85 instances over 12 issues

[N-01] `public` functions not called by the contract should be declared `external` instead

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

There are 8 instances of this issue:

```
File: contracts/governance/NounsDAOLogicV1.sol

174       function propose(
175           address[] memory targets,
176           uint256[] memory values,
177           string[] memory signatures,
178           bytes[] memory calldatas,
179           string memory description
180:      ) public returns (uint256) {

649       function _burnVetoPower() public {
650           // Check caller is pendingAdmin and pendingAdmin ≠
651:          require(msg.sender == vetoer, 'NounsDAO::_burnVetc

660:      function proposalThreshold() public view returns (uint
```

```
668:      function quorumVotes() public view returns (uint256) {
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV1.sol#L174-L180

```
File: contracts/governance/NounsDAOLogicV2.sol

184       function propose(
185           address[] memory targets,
186           uint256[] memory values,
187           string[] memory signatures,
188           bytes[] memory calldatas,
189           string memory description
190:      ) public returns (uint256) {

851       function _burnVetoPower() public {
852           // Check caller is pendingAdmin and pendingAdmin ≠
853:          require(msg.sender == vetoer, 'NounsDAO::_burnVetc

862:      function proposalThreshold() public view returns (uint

1002:     function maxQuorumVotes() public view returns (uint256
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L184-L190

[N-02] Non-assembly method available

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

There are 3 instances of this issue:

```
File: contracts/base/ERC721Checkpointable.sol
```

```
285:             chainId := chainid()
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/base/ERC721Checkpointable.sol#L285

```
File: contracts/governance/NounsDAOLogicV1.sol

679:             chainId := chainid()
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV1.sol#L679

```
File: contracts/governance/NounsDAOLogicV2.sol

1013:             chainId := chainid()
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L1013

[N-03] `2**<n> - 1` should be re-written as `type(uint<n>).max`

Earlier versions of solidity can use `uint<n>(-1)` instead. Expressions not including the `- 1` can often be re-written to accomodate the change (e.g. by using a `>` rather than a `>=`, which will also save some gas)

There are 2 instances of this issue:

```
File: contracts/base/ERC721Checkpointable.sol

254:         require(n < 2**32, errorMessage);
```

```
259:          require(n < 2**96, errorMessage);
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/base/ERC721Checkpointable.sol#L254

[N-04] `constant`s should be defined rather than using magic numbers

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

There are 8 instances of this issue:

```
File: contracts/base/ERC721Checkpointable.sol

/// @audit 32
254:          require(n < 2**32, errorMessage);

/// @audit 96
259:          require(n < 2**96, errorMessage);
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/base/ERC721Checkpointable.sol#L254

```
File: contracts/governance/NounsDAOLogicV1.sol

/// @audit 10000
673:          return (number * bps) / 10000;
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV1.sol#L673

```
File: contracts/governance/NounsDAOLogicV2.sol
```

```
/// @audit 10000
908:          uint256 againstVotesBPS = (10000 * againstVotes) /

/// @audit 1e6
909:          uint256 quorumAdjustmentBPS = (params.quorumCoeffi

/// @audit 10000
1007:         return (number * bps) / 10000;
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L908

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

/// @audit 0x20
98:                   revert(add(returnData, 0x20), returndatasi

/// @audit 0x40
113:              let free_mem_ptr := mload(0x40)
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOProxy.sol#L98

[N-05] Use a more recent version of solidity

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

There are 3 instances of this issue:

```
File: contracts/base/ERC721Checkpointable.sol

35:   pragma solidity ^0.8.6;
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/base

/ERC721Checkpointable.sol#L35

```
File: contracts/governance/NounsDAOLogicV1.sol

61:   pragma solidity ^0.8.6;
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV1.sol#L61

```
File: contracts/governance/NounsDAOLogicV2.sol

53:   pragma solidity ^0.8.6;
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L53

[N-06] Expressions for constant values such as a call to `keccak256()`, should use `immutable` rather than `constant`

While it doesn't save any gas because the compiler knows that developers often make this mistake, it's still best to use the right tool for the task at hand. There is a difference between `constant` variables and `immutable` variables, and they should each be used in their appropriate contexts. `constants` should be used for literal values written into the code, and `immutable` variables should be used for expressions, or values calculated in, or passed into the constructor.

There are 6 instances of this issue:

```
File: contracts/base/ERC721Checkpointable.sol

59        bytes32 public constant DOMAIN_TYPEHASH =
60:           keccak256('EIP712Domain(string name,uint256 chain]

63        bytes32 public constant DELEGATION_TYPEHASH =
```

```
64:           keccak256('Delegation(address delegatee,uint256 nc
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/base/ERC721Checkpointable.sol#L59-L60

```
File: contracts/governance/NounsDAOLogicV1.sol

97        bytes32 public constant DOMAIN_TYPEHASH =
98:           keccak256('EIP712Domain(string name,uint256 chainI

101:      bytes32 public constant BALLOT_TYPEHASH = keccak256('E
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV1.sol#L97-L98

```
File: contracts/governance/NounsDAOLogicV2.sol

101       bytes32 public constant DOMAIN_TYPEHASH =
102:          keccak256('EIP712Domain(string name,uint256 chainI

105:      bytes32 public constant BALLOT_TYPEHASH = keccak256('E
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L101-L102

[N-07] Constant redefined elsewhere

Consider defining in only one contract so that values cannot become out of sync when only one location is updated. A cheap way to store constants in a single location is to create an `internal constant` in a `library`. If the variable is a local cache of another contract's value, consider making the cache variable internal or private, which will require external users to query the contract with the source of truth, so that callers don't get out of sync.

There are 11 instances of this issue:

```
File: contracts/governance/NounsDAOLogicV2.sol

/// @audit seen in contracts/governance/NounsDAOLogicV1.sol
59:       string public constant name = 'Nouns DAO';

/// @audit seen in contracts/governance/NounsDAOLogicV1.sol
62:       uint256 public constant MIN_PROPOSAL_THRESHOLD_BPS = 1

/// @audit seen in contracts/governance/NounsDAOLogicV1.sol
65:       uint256 public constant MAX_PROPOSAL_THRESHOLD_BPS = 1

/// @audit seen in contracts/governance/NounsDAOLogicV1.sol
68:       uint256 public constant MIN_VOTING_PERIOD = 5_760; //

/// @audit seen in contracts/governance/NounsDAOLogicV1.sol
71:       uint256 public constant MAX_VOTING_PERIOD = 80_640; //

/// @audit seen in contracts/governance/NounsDAOLogicV1.sol
74:       uint256 public constant MIN_VOTING_DELAY = 1;

/// @audit seen in contracts/governance/NounsDAOLogicV1.sol
77:       uint256 public constant MAX_VOTING_DELAY = 40_320; //

/// @audit seen in contracts/governance/NounsDAOLogicV1.sol
89:       uint256 public constant MAX_QUORUM_VOTES_BPS = 2_000;

/// @audit seen in contracts/governance/NounsDAOLogicV1.sol
92:       uint256 public constant proposalMaxOperations = 10; //

/// @audit seen in contracts/governance/NounsDAOLogicV1.sol
101       bytes32 public constant DOMAIN_TYPEHASH =
102:          keccak256('EIP712Domain(string name,uint256 chainI

/// @audit seen in contracts/governance/NounsDAOLogicV1.sol
105:      bytes32 public constant BALLOT_TYPEHASH = keccak256('E
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L59

[N-08] Lines are too long

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

There are 7 instances of this issue:

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

156:       /// @notice The basis point number of votes in support

181:           /// @notice The number of votes in support of a pr

256:       /// @notice The basis point number of votes in support

281:           /// @notice The number of votes in support of a pr

375:           /// @notice The minimum number of votes in support
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOInterfaces.sol#L156

```
File: contracts/governance/NounsDAOLogicV1.sol

507:           /// @notice: Unlike GovernerBravo, votes are consi
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV1.sol#L507

```
File: contracts/governance/NounsDAOLogicV2.sol

599:           /// @notice: Unlike GovernerBravo, votes are consi
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L599

[N-09] Non-library/interface files should use fixed compiler versions, not floating ones

There are 4 instances of this issue:

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

33:   pragma solidity ^0.8.6;
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOInterfaces.sol#L33

```
File: contracts/governance/NounsDAOLogicV1.sol

61:   pragma solidity ^0.8.6;
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV1.sol#L61

```
File: contracts/governance/NounsDAOLogicV2.sol

53:   pragma solidity ^0.8.6;
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L53

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

```
36:   pragma solidity ^0.8.6;
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOProxy.sol#L36

[N-10] Event is missing `indexed` fields

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

There are 21 instances of this issue:

```
File: contracts/base/ERC721Checkpointable.sol

73:       event DelegateVotesChanged(address indexed delegate, u
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/base/ERC721Checkpointable.sol#L73

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

37        event ProposalCreated(
38            uint256 id,
39            address proposer,
40            address[] targets,
41            uint256[] values,
42            string[] signatures,
43            bytes[] calldatas,
44            uint256 startBlock,
45            uint256 endBlock,
46            string description
47:       );
```

```
50          event ProposalCreatedWithRequirements(
51              uint256 id,
52              address proposer,
53              address[] targets,
54              uint256[] values,
55              string[] signatures,
56              bytes[] calldatas,
57              uint256 startBlock,
58              uint256 endBlock,
59              uint256 proposalThreshold,
60              uint256 quorumVotes,
61              string description
62:         );

70:         event VoteCast(address indexed voter, uint256 proposal

73:         event ProposalCanceled(uint256 id);

76:         event ProposalQueued(uint256 id, uint256 eta);

79:         event ProposalExecuted(uint256 id);

82:         event ProposalVetoed(uint256 id);

85:         event VotingDelaySet(uint256 oldVotingDelay, uint256 n

88:         event VotingPeriodSet(uint256 oldVotingPeriod, uint256

91:         event NewImplementation(address oldImplementation, add

94:         event ProposalThresholdBPSSet(uint256 oldProposalThres

97:         event QuorumVotesBPSSet(uint256 oldQuorumVotesBPS, uin

100:        event NewPendingAdmin(address oldPendingAdmin, address

103:        event NewAdmin(address oldAdmin, address newAdmin);

106:        event NewVetoer(address oldVetoer, address newVetoer);

111:        event MinQuorumVotesBPSSet(uint16 oldMinQuorumVotesBPS

114:        event MaxQuorumVotesBPSSet(uint16 oldMaxQuorumVotesBPS

117:        event QuorumCoefficientSet(uint32 oldQuorumCoefficient
```

```
120:      event RefundableVote(address indexed voter, uint256 re

123:      event Withdraw(uint256 amount, bool sent);
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOInterfaces.sol#L37-L47

[N-11] Not using the named return variables anywhere in the function is confusing

Consider changing the variable to be an unnamed one

There are 8 instances of this issue:

```
File: contracts/governance/NounsDAOLogicV1.sol

/// @audit targets
/// @audit values
/// @audit signatures
/// @audit calldatas
392       function getActions(uint256 proposalId)
393           external
394           view
395           returns (
396               address[] memory targets,
397               uint256[] memory values,
398               string[] memory signatures,
399:              bytes[] memory calldatas
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV1.sol#L392-L399

```
File: contracts/governance/NounsDAOLogicV2.sol

/// @audit targets
/// @audit values
/// @audit signatures
```

```
/// @audit calldatas
403       function getActions(uint256 proposalId)
404           external
405           view
406           returns (
407               address[] memory targets,
408               uint256[] memory values,
409               string[] memory signatures,
410:              bytes[] memory calldatas
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L403-L410

[N-12] Typos

There are 4 instances of this issue:

```
File: /contracts/governance/NounsDAOLogicV1.sol

/// @audit contructor
104:      * @notice Used to initialize the contract during deleg

/// @audit priviledges
646:      * @notice Burns veto priviledges
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV1.sol#L104

```
File: /contracts/governance/NounsDAOLogicV2.sol

/// @audit contructor
115:      * @notice Used to initialize the contract during deleg

/// @audit priviledges
848:      * @notice Burns veto priviledges
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L115

Gas Optimizations

For this contest, 126 reports were submitted by wardens detailing gas optimizations. The report highlighted below by IIIIIII received the top score from the judge.

The following wardens also submitted reports: JohnSmith, m_Rassska, Dravee, 0xDjango, 0x1f8b, cRat1st0s, 0xSmartContract, Aymen0909, oyc_109, ReyAdmirado, Bnke0x0, pfapostol, Rolezn, robee, joestakey, TomJ, erictee, mics, ajtra, 0xkatana, ElKu, martin, Sm4rty, Junnon, Deivitto, ch0bu, c3phas, fatherOfBlocks, ladboy233, sikorico, Olivierdem, Tomo, 0xNazgul, ret2basic, _141345_, prasantgupta52, gogo, GalloDaSballo, JC, rbserver, durianSausage, DimitarDimitrov, samruna, LeoS, jag, carlitox477, Ruhum, Shishigami, 0xbepresent, Ben, ignacio, SaharAP, 0x040, sach1r0, BipinSah, bulej93, lucacez, CertoraInc, sryysryy, rvierdiiev, exolorkistis, zishansami, saian, Tomio, Rohan16, rfa, Fitraldys, hyh, pauliax, brgltd, natzuu, Chom, medikko, lukris02, delfin454000, d3e4, EthLedger, DevABDee, Saintcode_, Lambda, djxploit, rokinot, shr1ftyy, CodingNameKiki, mrpathfindr, Noah3o6, rotcivegaf, ak1, Respx, simon135, RaymondFam, karanctf, _Adam, shark, GimelSec, catchup, 0xc0ffEE, Waze, 0xNineDec, KIntern_NA, SooYa, Guardian, Ch_301, Amithuddar, SerMyVillage, 2997ms, newfork01, RoiEvenHaim, Polandia94, tay054, Yiko, Bjorn_bug, bobirichman, ACai, seyni, francoHacker, Randyyy, R2, Diraco, Funen, IgnacioB, tnevler, wagmi, peritoflores, *and* a12jmx.

Summary

	Issue	Instances
[G-01]	State checks unnecessarily re-fetch `Proposal` s	5
[G-02]	Multiple `address` /ID mappings can be combined into a single `mapping` of an `address` /ID to a `struct` , where appropriate	1
[G-03]	Structs can be packed into fewer storage slots	3

	Issue	Instances
[G-04]	Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas	10
[G-05]	Using `storage` instead of `memory` for structs/arrays saves gas	1
[G-06]	State variables should be cached in stack variables rather than re-reading them from storage	11
[G-07]	Multiple accesses of a mapping/array should use a local variable cache	2
[G-08]	`internal` functions only called once can be inlined to save gas	7
[G-09]	Add `unchecked {}` for subtractions where the operands cannot underflow because of a previous `require()` or `if` -statement	1
[G-10]	`<array>.length` should not be looked up in every loop of a `for` -loop	8
[G-11]	`++i` / `i++` should be `unchecked{++i}` / `unchecked{i++}` when it is not possible for them to overflow, as is the case when used in `for` - and `while` -loops	8
[G-12]	`require()` / `revert()` strings longer than 32 bytes cost extra gas	86
[G-13]	Optimize names to save gas	5
[G-14]	Use a more recent version of solidity	1
[G-15]	`++i` costs less gas than `i++` , especially when it's used in `for` -loops (`--i` / `i--` too)	10
[G-16]	Splitting `require()` statements that use `&&` saves gas	19
[G-17]	Usage of `uints` / `ints` smaller than 32 bytes (256 bits) incurs overhead	1
[G-18]	Using `private` rather than `public` for constants, saves gas	31
[G-19]	Don't compare boolean expressions to boolean literals	2
[G-20]	Division by two should use bit shifting	2

	Issue	Instances
[G-21]	`require()` or `revert()` statements that check input arguments should be at the top of the function	3
[G-22]	Empty blocks should be removed or emit something	1
[G-23]	Use custom errors rather than `revert()` / `require()` strings to save gas	95

Total: 313 instances over 23 issues

[G-01] State checks unnecessarily re-fetch `Proposal` s

Every call to `state()` fetches the `Proposal` storage variable, which is fetched again immediately afterwards by the caller. If instead there were a version of `state()` that took in a `Proposal storage` variable, the proposal could be fetched only once, saving the gas of the mapping lookup

There are 5 instances of this issue. (For in-depth details on this and all further gas optimizations with multiple instances, please see the warden's full report.)

[G-02] Multiple `address` /ID mappings can be combined into a single `mapping` of an `address` /ID to a `struct` , where appropriate

Saves a storage slot for the mapping. Depending on the circumstances and sizes of types, can avoid a Gsset (**20000 gas**) per mapping combined. Reads and subsequent writes can also be cheaper when a function requires both values and they both fit in the same storage slot. Finally, if both fields are accessed in the same function, can save **~42 gas per access** due to not having to recalculate the key's keccak256 hash (Gkeccak256 - 30 gas) and that calculation's associated stack operations.

There is 1 instance of this issue:

File: contracts/base/ERC721Checkpointable.sol

```
53        mapping(address => mapping(uint32 => Checkpoint)) publ
54
55        /// @notice The number of checkpoints for each account
56:       mapping(address => uint32) public numCheckpoints;
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/base/ERC721Checkpointable.sol#L53-L56

[G-03] Structs can be packed into fewer storage slots

Each slot saved can avoid an extra Gsset (**20000 gas**) for the first setting of the struct. Subsequent reads as well as writes have smaller gas savings

There are 3 instances of this issue.

[G-04] Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas

When a function with a `memory` array is called externally, the `abi.decode()` step has to use a for-loop to copy each index of the `calldata` to the `memory` index. **Each iteration of this for-loop costs at least 60 gas** (i.e. `60 * <mem_array>.length`). Using `calldata` directly, obliviates the need for such a loop in the contract code and runtime execution. Note that even if an interface defines a function as having `memory` arguments, it's still valid for implementation contracs to use `calldata` arguments instead.

If the array is passed to an `internal` function which passes the array to another internal function where the array is modified and therefore `memory` is used in the `external` call, it's still more gass-efficient to use `calldata` when the `external` function uses modifiers, since the modifiers may prevent the internal functions from being called. Structs have the same overhead as an array of length one

Note that I've also flagged instances where the function is `public` but can be marked as `external` since it's not called by the contract, and cases where a constructor is involved

There are 10 instances of this issue.

[G-05] Using `storage` instead of `memory` for structs/arrays saves gas

When fetching data from a storage location, assigning the data to a `memory` variable causes all fields of the struct/array to be read from storage, which incurs a Gcoldsload (**2100 gas**) for *each* field of the struct/array. If the fields are read from the new memory variable, they incur an additional `MLOAD` rather than a cheap stack read. Instead of declearing the variable with the `memory` keyword, declaring the variable with the `storage` keyword and caching any fields that need to be re-read in stack variables, will be much cheaper, only incuring the Gcoldsload for the fields actually read. The only time it makes sense to read the whole struct/array into a `memory` variable, is if the full struct/array is being returned by the function, is being passed to a function that requires `memory`, or if the array/struct is being read from another `memory` array/struct

There is 1 instance of this issue:

```
File: contracts/governance/NounsDAOLogicV2.sol

952:                 DynamicQuorumParamsCheckpoint memory cp = quor
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L952

[G-06] State variables should be cached in stack variables rather than re-reading them from storage

The instances below point to the second+ access of a state variable within a function. Caching of a state variable replace 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.

There are 11 instances of this issue.

[G-07] Multiple accesses of a mapping/array should use a local variable cache

The instances below point to the second+ access of a value inside a mapping/array, within a function. Caching a mapping's value in a local `storage` or `calldata` variable when the value is accessed multiple times, saves **~42 gas per access** due to not having to recalculate the key's keccak256 hash (Gkeccak256 - **30 gas**) and that calculation's associated stack operations. Caching an array's struct avoids recalculating the array offsets into memory/calldata

There are 2 instances of this issue.

[G-08] `internal` functions only called once can be inlined to save gas

Not inlining costs **20 to 40 gas** because of two extra `JUMP` instructions and additional stack operations needed for function calls.

There are 7 instances of this issue.

[G-09] Add `unchecked {}` for subtractions where the operands cannot underflow because of a previous `require()` or `if` -statement

`require(a <= b); x = b - a` **=>** `require(a <= b); unchecked { x = b - a }`

There is 1 instance of this issue:

```
File: contracts/base/ERC721Checkpointable.sol

/// @audit require() on line 278
279:           return a - b;
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/base

/ERC721Checkpointable.sol#L279

[G-10] `<array>.length` should not be looked up in every loop of a `for` -loop

The overheads outlined below are *PER LOOP*, excluding the first loop

- storage arrays incur a Gwarmaccess (**100 gas**)
- memory arrays use `MLOAD` (**3 gas**)
- calldata arrays use `CALLDATALOAD` (**3 gas**)

Caching the length changes each of these to a `DUP<N>` (**3 gas**), and gets rid of the extra `DUP<N>` needed to store the stack offset

There are 8 instances of this issue.

[G-11] `++i` / `i++` should be `unchecked{++i}` / `unchecked{i++}` when it is not possible for them to overflow, as is the case when used in `for` - and `while` -loops

The `unchecked` keyword is new in solidity version 0.8.0, so this only applies to that version or higher, which these instances are. This saves **30-40 gas** per loop

There are 8 instances of this issue.

[G-12] `require()` / `revert()` strings longer than 32 bytes cost extra gas

Each extra memory word of bytes past the original 32 incurs an MSTORE which costs **3 gas**

There are 86 instances of this issue.

[G-13] Optimize names to save gas

`public` / `external` function names and `public` member variable names can be optimized to save gas. See this link for an example of how it works. Below are the interfaces/abstract contracts that can be optimized so that the most frequently-called functions use the least amount of gas possible during method lookup. Method IDs that have two leading zero bytes can save **128 gas** each during deployment, and renaming functions to have lower method IDs will save **22 gas** per call, per sorted position shifted

There are 5 instances of this issue.

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

Use a solidity version of at least 0.8.2 to get simple compiler automatic inlining Use a solidity version of at least 0.8.3 to get better struct packing and cheaper multiple storage reads Use a solidity version of at least 0.8.4 to get custom errors, which are cheaper at deployment than `revert()/require()` strings Use a solidity version of at least 0.8.10 to have external calls skip contract existence checks if the external call has a return value

There is 1 instance of this issue:

```
File: contracts/base/ERC721Enumerable.sol

28:   pragma solidity ^0.8.0;
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/base/ERC721Enumerable.sol#L28

[G-15] `++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i` / `i--` too)

Saves **5 gas per loop**

There are 10 instances of this issue.

[G-16] Splitting `require()` statements that use `&&` saves gas

See [this issue](#) which describes the fact that there is a larger deployment gas cost, but with enough runtime calls, the change ends up being cheaper by **3 gas**

There are 19 instances of this issue.

[G-17] Usage of `uints` / `ints` smaller than 32 bytes (256 bits) incurs overhead

> When using elements that are smaller than 32 bytes, your contract's gas usage may be higher. This is because the EVM operates on 32 bytes at a time. Therefore, if the element is smaller than that, the EVM must use more operations in order to reduce the size of the element from 32 bytes to the desired size.

https://docs.soliditylang.org/en/v0.8.11/internals/layout_in_storage.html Each operation involving a `uint8` costs an extra 22-28 gas (depending on whether the other operand is also a variable of type `uint8`) as compared to ones involving `uint256`, due to the compiler having to clear the higher bits of the memory word before operating on the `uint8`, as well as the associated stack operations of doing so. Use a larger size then downcast where needed

There is 1 instance of this issue:

```
File: contracts/base/ERC721Checkpointable.sol

/// @audit uint32 upper
191:                 upper = center - 1;
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/base/ERC721Checkpointable.sol#L191

[G-18] Using `private` rather than `public` for constants, saves gas

If needed, the values can be read from the verified contract source code, or if there are multiple values there can be a single getter function that [returns a tuple](#) of the values of all currently-public constants. Saves **3406-3606 gas** in deployment gas due to the compiler not having to create non-payable getter functions for deployment calldata, not having to store the bytes of the value outside of where it's used, and not adding another entry to the method ID table

There are 31 instances of this issue.

[G-19] Don't compare boolean expressions to boolean literals

`if (<x> == true)` **=>** `if (<x>)` , `if (<x> == false)` **=>** `if (!<x>)`

There are 2 instances of this issue.

[G-20] Division by two should use bit shifting

`<x> / 2` is the same as `<x> >> 1` . While the compiler uses the `SHR` opcode to accomplish both, the version that uses division incurs an overhead of [20 gas](#) due to `JUMP` s to and from a compiler utility function that introduces checks which can be avoided by using `unchecked {}` around the division by two

There are 2 instances of this issue.

[G-21] `require()` or `revert()` statements that check input arguments should be at the top of the function

Checks that involve constants should come before checks that involve state variables, function calls, and calculations. By doing these checks first, the function is able to revert before wasting a Gcoldsload (**2100 gas***) in a function that may ultimately revert in the unhappy case.

There are 3 instances of this issue.

[G-22] Empty blocks should be removed or emit something

The code should be refactored such that they no longer exist, or the block should do something useful, such as emitting an event or reverting. If the contract is meant to

be extended, the contract should be `abstract` and the function signatures be added without any default implementation. If the block is an empty `if` -statement block to avoid doing subsequent checks in the else-if/else conditions, the else-if/else conditions should be nested under the negation of the if-statement, because they involve different classes of checks, which may lead to the introduction of errors when the code is later modified (`if(x){}else if(y){...}else{...}` => `if(!x){if(y){...}else{...}}`). Empty `receive()` / `fallback() payable` functions that are not used, can be removed to save deployment gas.

There is 1 instance of this issue:

```
File: contracts/governance/NounsDAOLogicV2.sol

1030:     receive() external payable {}
```

https://github.com/code-423n4/2022-08-nounsdao/blob/45411325ec14c6d747b999a40367d3c5109b5a89/contracts/governance/NounsDAOLogicV2.sol#L1030

[G-23] Use custom errors rather than `revert()` / `require()` strings to save gas

Custom errors are available from solidity version 0.8.4. Custom errors save ~50 gas each time they're hit by avoiding having to allocate and store the revert string. Not defining the strings also save deployment gas

There are 95 instances of this issue.

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