Nouns DAO
Findings & Analysis Report

2023-07-31

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 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 outlined in this document, C4 conducted an analysis of the Nouns DAO smart contract system written in Solidity. The audit took place between July 3—July 13, 2023.

Wardens

36 Wardens contributed reports to the Nouns DAO:

1. 0xA5DF
2. 0xAnah
3. 0xG0P1
4. 0xMilenov
5. 0xSmartContract
6. 0xnev
7. Aymen0909
8. Bauchibred

9. Emmanuel
10. JCN
11. K42
12. Kaysoft
13. Matin
14. MohammedRizwan
15. Raihan
16. SAQ
17. SM3_SS
18. c3phas
19. cccz
20. codegpt
21. descharre
22. dharma09
23. fatherOfBlocks
24. flutter_developer
25. hunter_w3b
26. iglyx
27. ihtishamsudo
28. jasonxiale
29. klau5
30. koxuan
31. kutugu
32. nadin
33. naman1778
34. petrichor
35. said
36. shark

This audit was judged by gzeon.

Final report assembled by liveactionllama.

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 14 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 15 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 repository, and is composed of 33 smart contracts written in the Solidity programming language and includes 9,098 lines of Solidity code.

Severity Criteria

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

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

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

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

High Risk Findings (1)

[H-01] User can steal tokens by using duplicated ERC20 tokens as parameter in `NounsDAOLogicV1Fork.quit`

Submitted by jasonxiale, also found by iglyx, 0xA5DF, said, shark, and 0xG0P1

Calling NounsDAOLogicV1Fork.quit by using dupliated ERC20 tokens, malicious user can gain more ERC20 tokens than he/she is supposed to, even drain all ERC20 tokens.

Proof of Concept

In function, NounsDAOLogicV1Fork.quit, `erc20TokensToInclude` is used to specified tokens a user wants to get, but since the function doesn't verify if `erc20TokensToInclude` contains dupliated tokens, it's possible that a malicious user calls the function by specify the ERC20 more than once to get more share tokens.

```
    function quit(uint256[] calldata tokenIds, address[] memory
        // check that erc20TokensToInclude is a subset of `erc2
        address[] memory erc20TokensToIncludeInQuit_ = erc20Toke
        for (uint256 i = 0; i < erc20TokensToInclude.length; i+
            if (!isAddressIn(erc20TokensToInclude[i], erc20Toke
                revert TokensMustBeASubsetOfWhitelistedTokens();
            }
        }

        quitInternal(tokenIds, erc20TokensToInclude);
    }

    function quitInternal(uint256[] calldata tokenIds, address[]
        checkGovernanceActive();

        uint256 totalSupply = adjustedTotalSupply();

        for (uint256 i = 0; i < tokenIds.length; i++) {
            nouns.transferFrom(msg.sender, address(timelock), to
        }

        uint256[] memory balancesToSend = new uint256[](erc20To
        // Capture balances to send before actually sending then
        uint256 ethToSend = (address(timelock).balance * tokenIc
        for (uint256 i = 0; i < erc20TokensToInclude.length; i++
            IERC20 erc20token = IERC20(erc20TokensToInclude[i]);
            balancesToSend[i] = (erc20token.balanceOf(address(ti
        }

        // Send ETH and ERC20 tokens
        timelock.sendETH(payable(msg.sender), ethToSend);
        for (uint256 i = 0; i < erc20TokensToInclude.length; i++
            if (balancesToSend[i] > 0) {
                timelock.sendERC20(msg.sender, erc20TokensToIncl
            }
        }

        emit Quit(msg.sender, tokenIds);
    }
```

Add the following code in test/foundry/governance/fork/NounsDAOLogicV1Fork.t.sol file `NounsDAOLogicV1Fork_Quit_Test` contract, and run `forge test --ffi --mt test_quit_allowsChoosingErc20TokensToIncludeTwice`.

```
    function test_quit_allowsChoosingErc20TokensToIncludeTwice()
        vm.prank(quitter);
        address[] memory tokensToInclude = new address[](3);
        //****************************
        // specify token2 three times
        //****************************
        tokensToInclude[0] = address(token2);
        tokensToInclude[1] = address(token2);
        tokensToInclude[2] = address(token2);
        dao.quit(quitterTokens, tokensToInclude);

        assertEq(quitter.balance, 24 ether);
        assertEq(token1.balanceOf(quitter), 0);
        //****************************
        // get 3 time tokens
        //****************************
        assertEq(token2.balanceOf(quitter), 3 * (TOKEN2_BALANCE
     }
```

Tools Used

VS

Recommended Mitigation Steps

By using function `checkForDuplicates` to prevent the issue

```
--- NounsDAOLogicV1Fork.sol     2023-07-12 21:32:56.925848531 +0
+++ NounsDAOLogicV1ForkNew.sol  2023-07-12 21:32:34.006158294 +0
@@ -203,8 +203,9 @@
         quitInternal(tokenIds, erc20TokensToIncludeInQuit);
     }

-    function quit(uint256[] calldata tokenIds, address[] memory
+    function quit(uint256[] calldata tokenIds, address[] memory
         // check that erc20TokensToInclude is a subset of `erc2
+        checkForDuplicates(erc20tokenstoinclude);
         address[] memory erc20TokensToIncludeInQuit_ = erc20Tok
         for (uint256 i = 0; i < erc20TokensToInclude.length; i+
             if (!isAddressIn(erc20TokensToInclude[i], erc20Toke
```

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

> Fix PR: https://github.com/nounsDAO/nouns-monorepo/pull/762

[gzeon (judge) increased severity to High](#)

Medium Risk Findings (3)

[M-01] `cancelSig` will not completely cancel signatures due to malleability vulnerabilities

Submitted by [kutugu](#)

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/NounsDAOV3Proposals.sol#L270-L275
https://github.com/nounsDAO/nouns-

monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/NounsDAOV3Proposals.sol#L983

The current version of openzeppelin contracts has a high risk of vulnerability about signature malleability attack: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/3610.

So if the signer only cancel one signature, the malicious proposer can still extend a fully valid signature through the previous signature to pass the proposal.

Proof of Concept

▶ Details

Tools Used

Foundry

Recommended Mitigation Steps

Update openzeppelin contracts to the new version.

eladmallel (Nouns DAO) confirmed, but disagreed with severity and commented:

> Fix PR here: https://github.com/nounsDAO/nouns-monorepo/pull/761

> However, think severity should not be high. The worst case here is a signature abuse leads to a proposal going on chain, still subject to the proposal lifecycle, including quorum and voting.

davidbrai (Nouns DAO) commented:

> Another point regarding severity:
> The signer can also move their tokens to another address as a way to make the previous signature not useful.

gzeon (judge) decreased severity to Low and commented:

> Downgrading to Low since no asset will be at risk and require a user error.

gzeon (judge) increased severity to Medium and commented:

> It is worth to note this is atypical in Code4rena judging, and should not be considered as a precedence for future contests. Signature malleability, or outdated OZ dependency are generally considered as out-of-scope in C4 contests as they are covered by the bot report. This report is special in the sense that while the project already used the recommended OZ ECDSA library, the specific version they used contained a bug that allow malleability, which the warden provided a POC with meaningful impact. I am keeping this as Medium risk for the above reason and sponsor opinion.

[M-02] If DAO updates `forkEscrow` before `forkThreshold` is reached, the user's escrowed Nouns will be lost

Submitted by cccz

During the escrow period, users can escrow to or withdraw from forkEscrow their Nouns.

During the escrow period, proposals can be executed.

```
    function withdrawFromForkEscrow(NounsDAOStorageV3.StorageV3
        if (isForkPeriodActive(ds)) revert ForkPeriodActive();

        INounsDAOForkEscrow forkEscrow = ds.forkEscrow;
        forkEscrow.returnTokensToOwner(msg.sender, tokenIds);

        emit WithdrawFromForkEscrow(forkEscrow.forkId(), msg.ser
    }
```

Since withdrawFromForkEscrow will only call the returnTokensToOwner function of ds.forkEscrow, and returnTokensToOwner is only allowed to be called by DAO.

If, during the escrow period, ds.forkEscrow is changed by the proposal's call to _setForkEscrow, then the user's escrowed Nouns will not be withdrawn by withdrawFromForkEscrow.

```
    function returnTokensToOwner(address owner, uint256[] callda
        for (uint256 i = 0; i < tokenIds.length; i++) {
            if (currentOwnerOf(tokenIds[i]) != owner) revert Not

            nounsToken.transferFrom(address(this), owner, tokenI
            escrowedTokensByForkId[forkId][tokenIds[i]] = addres
        }

        numTokensInEscrow -= tokenIds.length;
    }
```

Consider that some Nouners is voting on a proposal that would change ds.forkEscrow.
There are some escrowed Nouns in forkEscrow (some Nouners may choose to always escrow their Nouns to avoid missing fork).
The proposal is executed, ds.forkEscrow is updated, and the escrowed Nouns cannot be withdrawn.

Proof of Concept

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/fork/NounsDAOV3Fork.sol#L95-L102
https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/fork/NounsDAOForkEscrow.sol#L116-L125
https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/NounsDAOV3Admin.sol#L527-L531

Recommended Mitigation Steps

Consider allowing the user to call forkEscrow.returnTokensToOwner directly to withdraw escrowed Nouns, and need to move isForkPeriodActive from withdrawFromForkEscrow to returnTokensToOwner.

eladmallel (Nouns DAO) acknowledged

[M-03] `NounsDAOV3Proposals.cancel()` should allow to cancel the proposal of the Expired state

Submitted by cccz

cancel() does not allow to cancel proposals in the final states Canceled/Defeated/Expired/Executed/Vetoed.

```
    function cancel(NounsDAOStorageV3.StorageV3 storage ds, uint
        NounsDAOStorageV3.ProposalState proposalState = stateInt
        if (
            proposalState == NounsDAOStorageV3.ProposalState.Car
            proposalState == NounsDAOStorageV3.ProposalState.Def
            proposalState == NounsDAOStorageV3.ProposalState.Exp
            proposalState == NounsDAOStorageV3.ProposalState.Exe
            proposalState == NounsDAOStorageV3.ProposalState.Vet
        ) {
            revert CantCancelProposalAtFinalState();
        }
```

The Canceled/Executed/Vetoed states are final because they cannot be changed once they are set.

The Defeated state is also a final state because no new votes will be cast (`stateInternal()` may return Defeated only if the `objectionPeriodEndBlock` is passed).

But the Expired state depends on the `GRACE_PERIOD` of the timelock, and `GRACE_PERIOD` may be changed due to upgrades. Once the `GRACE_PERIOD` of the timelock is changed, the state of the proposal may also be changed, so Expired is not the final state.

```
        } else if (block.timestamp >= proposal.eta + getProposal
            return NounsDAOStorageV3.ProposalState.Expired;
        } else {
            return NounsDAOStorageV3.ProposalState.Queued;
```

Consider the following scenario:

- Alice submits proposal A to stake 20,000 ETH to a DEFI protocol, and it is successfully passed, but it cannot be executed because there is now only 15,000 ETH in the timelock (consumed by other proposals), and then proposal A expires.
- The DEFI protocol has been hacked or rug-pulled.
- Now proposal B is about to be executed to upgrade the timelock and extend `GRACE_PERIOD` (e.g., `GRACE_PERIOD` is extended by 7 days from V1 to V2).
- Alice wants to cancel Proposal A, but it cannot be canceled because it is in Expired state.
- Proposal B is executed, causing Proposal A to change from Expired to Queued.
- The malicious user sends 5000 ETH to the timelock and immediately executes Proposal A to send 20000 ETH to the hacked protocol.

Proof of Concept

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/NounsDAOV3Proposals.sol#L571-L581

Recommended Mitigation Steps

Consider adding a proposal expiration time field in the Proposal structure.

```
    function queue(NounsDAOStorageV3.StorageV3 storage ds, uint2
        require(
            stateInternal(ds, proposalId) == NounsDAOStorageV3.I
            'NounsDAO::queue: proposal can only be queued if it
        );
        NounsDAOStorageV3.Proposal storage proposal = ds._propos
        INounsDAOExecutor timelock = getProposalTimelock(ds, pro
        uint256 eta = block.timestamp + timelock.delay();
        for (uint256 i = 0; i < proposal.targets.length; i++) {
            queueOrRevertInternal(
                timelock,
                proposal.targets[i],
                proposal.values[i],
                proposal.signatures[i],
                proposal.calldatas[i],
                eta
            );
            }
            proposal.eta = eta;
+           proposal.exp = eta + timelock.GRACE_PERIOD();
...
-           } else if (block.timestamp >= proposal.eta + getProposal
+           } else if (block.timestamp >= proposal.exp) {
                return NounsDAOStorageV3.ProposalState.Expired;
```

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

> Agree, it's possible due to a change in executor's grace period to move from Expired back to Queued.
> However, since a grace period change is a rare event, we think this is very low priority and we won't fix.

[gzeon (judge) decreased severity to Low/Non-Critical]()

[eladmallel (Nouns DAO) commented]():

> We think it would be great to include this issue in the report (at medium severity).

[gzeon (judge) commented]():

> @eladmallel - Changing the `GRACE_PERIOD` is an admin change, which besides misconfiguration is out-of-scope, it is as you described is a rare event. Having a malicious proposal which is passed that got expired is also a rare event. Having a changed `GRACE_PERIOD` that just long enough to make such a malicious proposal become queued is a very rare event, assuming governance is not completely compromised already.

> That said, I am ok with this being Medium risk since this is clearly in scope + can be Medium risk with some assumption (tho extreme imo but is subjective), and I would recommend for a fix accordingly. Please let me know if that's what you want, thanks!

[eladmallel (Nouns DAO) commented]():

> Thank you @gzeon.
> We all agree the odds of the risk materializing is low, we just felt like this was a

> nice find, and honestly mostly motivated by wanting the warden who found this to have a win :)

> It's not a deal breaker for us if it's in the report or not, just wanted to express our preference.

> Thank you for sharing more of your thinking, it's helpful!

cccz (warden) commented:

> Low Likelihood + High Severity is generally considered Medium, which is an edge case that fits the medium risk.
> Another thing I would say is that the proposal doesn't need to be malicious, as I said in the attack scenario where the proposal is normal but expires due to inability to execute for other reasons (contract balance insufficient, etc.).

> *Changing the `GRACE_PERIOD` is an admin change, which besides misconfiguration is out-of-scope, it is as you described is a rare event. Having a malicious proposal which is passed that got expired is also a rare event. Having a changed `GRACE_PERIOD` that just long enough to make such a malicious proposal become queued is a very rare event, assuming governance is not completely compromised already.*

gzeon (judge) increased severity to Medium and commented:

> @cccz - True, but this is also marginally out-of-scope since an admin action is required, and one may argue it is a misconfiguration if you increase `GRACE_PERIOD` so much that it revive some old passed buggy proposal.

> But given this is marginal and on sponsor's recommendation, I will upgrade this to Medium.

Low Risk and Non-Critical Issues

For this audit, 12 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **shark** received the top score from the judge.

The following wardens also submitted reports: Kaysoft, codegpt, MohammedRizwan, klau5, nadin, 0xMilenov, ihtishamsudo, descharre, fatherOfBlocks, koxuan, *and* Bauchibred.

[01] Reserve price not fully taken care of

It is possible for an active auction to close at a price lower than the newly increased reserve price. This is undesirable especially when preventing a Noun auctioned off at the lower than expected price could be out of control in a bear market. Consider adding a check alleging that the contract balance needing to exceed the reserve price. Else, the last bidder will be refunded prior to having the Noun burned. Here's a refactored code logic that will take care of the suggestion.

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/fork/newdao/NounsAuctionHouseFork.sol#L236-L256

```
    /**
     * @notice Settle an auction, finalizing the bid and paying out
     * @dev If there are no bids, the Noun is burned.
     */
    function _settleAuction() internal {
        INounsAuctionHouse.Auction memory _auction = auction;

        require(_auction.startTime != 0, "Auction hasn't begun");
        require(!_auction.settled, 'Auction has already been settlec
        require(block.timestamp >= _auction.endTime, "Auction hasn't

        auction.settled = true;

        // Check if contract balance is greater than reserve price
        if (address(this).balance < reservePrice) {
            // If contract balance is less than reserve price, refur
            if (_auction.bidder != address(0)) {
                _safeTransferETHWithFallback(_auction.bidder, _aucti
            }

            // And then burn the Noun
            nouns.burn(_auction.nounId);
        } else {
            if (_auction.bidder == address(0)) {
                nouns.burn(_auction.nounId);
            } else {
                nouns.transferFrom(address(this), _auction.bidder, _
            }

            if (_auction.amount > 0) {
                _safeTransferETHWithFallback(owner(), _auction.amour
            }
        }

        emit AuctionSettled(_auction.nounId, _auction.bidder, _aucti
    }
```

[02] Code and comment mismatch (V2 Only)

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/NounsDAOLogicV2.sol#L86

```
        @ audit 4,000 should be changed to 6,000
        uint256 public constant MAX_QUORUM_VOTES_BPS_UPPER_BOUND = (
```

[03] Spelling errors

There are numerous instances throughout the codebase in different contracts. Here's just one of the specific instances:

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/NounsDAOLogicV2.sol#L61

```
        @ audit setable should be changed to settable
        /// @notice The minimum setable proposal threshold
```

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/NounsDAOLogicV2.sol#L218

```
            @ audit arity should be changed to parity
            'NounsDAO::propose: proposal function information ar
```

There are numerous instances throughout the codebase in different contracts. Here's just one of the specific instances:

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/NounsDAOLogicV2.sol#L901

```
       @ audit priviledges should be changed to privileges
       * @notice Burns veto priviledges
```

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/NounsDAOLogicV3.sol#L19

```
   @ audit NounsDAOLogicV2.sol should be changed to NounsDAOLogicV3
   // NounsDAOLogicV2.sol is a modified version of Compound Lab's C
```

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/NounsDAOInterfaces.sol#L217

```
      @ audit the during of should be omitted
      /// @notice Emitted when the during of the forking period is
```

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/fork/newdao/token/base/ERC721CheckpointableUpgradeable.sol#L80-L84

```
      @ audit thats should be changed to that's
```

```
    /// @notice An event thats emitted when an account changes i
    event DelegateChanged(address indexed delegator, address inc

    @ audit thats should be changed to that's
    /// @notice An event thats emitted when a delegate account's
    event DelegateVotesChanged(address indexed delegate, uint25(
```

[04] Wrong adoption of block time (V2 Only)

The following voting period constants are assuming 9.6 instead of 12 seconds per block. Depending on the sensitivity of lower and upper ranges desired, these may limit or shift the intended settable voting periods. For instance, using the supposed 12 second per block convention, the minimum and maximum settable voting periods should respectively be `7_200` and `100_800`.

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/NounsDAOLogicV2.sol#L67-L71

```
    /// @notice The minimum setable voting period
    uint256 public constant MIN_VOTING_PERIOD = 5_760; // About

    /// @notice The max setable voting period
    uint256 public constant MAX_VOTING_PERIOD = 80_640; // About
```

[05] `MIN_PROPOSAL_THRESHOLD_BPS` is too low a value

In NounsDAOLogicV2.sol, NounsDAOV3Admin.sol, and NounsDAOLogicV1Fork.sol, the minimum proposal threshold can be set as low as 1 basis point.

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/NounsDAOLogicV2.sol#L61-L62
https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/NounsDAOV3Admin.sol#L111-L112
https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/noun

s-
contracts/contracts/governance/fork/newdao/governance/NounsDAOLogicV1Fork
.sol#L118-L119

```
    /// @notice The minimum setable proposal threshold
    uint256 public constant MIN_PROPOSAL_THRESHOLD_BPS = 1; // 1
```

This could pose a precision issue even if the total supply of Nouns tokens is already in its three digits. Apparently, a proposal threshold determined via the following two functions could return zero, e.g. `(1 * 720) / 10000` yields zero due to truncation, i.e. the numerator smaller than the denominator.

https://github.com/nounsDAO/nouns-
monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/noun
s-contracts/contracts/governance/NounsDAOLogicV2.sol#L921-L923

```
    function proposalThreshold() public view returns (uint256) {
        return bps2Uint(proposalThresholdBPS, nouns.totalSupply(
    }
```

https://github.com/nounsDAO/nouns-
monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/noun
s-contracts/contracts/governance/NounsDAOLogicV2.sol#L1066-L1068

```
    function bps2Uint(uint256 bps, uint256 number) internal pure
        return (number * bps) / 10000;
    }
```

[06] V3 upgrade could depreciate token value due to oversupply

The forking feature, though rarely happened as documented in the FAQs, could potentially affect the Nouns token value. This is because:

- a new fork will have its own DAO tokens daily generated and auctioned off through a new Auction House.

- new Nouns tokens claimed through escrow and during forking period will not have the original Nouns tokens burned. They are transferred to the original treasury or elsewhere as the DAO deems fit.

With a new fork competing with the original DAO for the daily auction, it will likely diverge the amount of ETH intended to go into bidding for the daily new NFTs at opposing ends. The situation could be worse in the far future if more forks were to transpire.

[07] Nouns fork may not efficiently remedy a bad situation

The `20%` threshold, as documented in the FAQs for instance, `800 * 0.2 = 160` of Nouns tokens, is not a small number comparatively in terms of the market cap. This translates to approximately `160 * 30 ETH * $2,000` almost equivalent to 10 million worth of USDC. For members wishing to dodge bad/undesirable proposals that aren't going to be vetoed, it's likely this will not materialize where the proposals get executed long before the threshold could be met to initiate a fork.

Consider conditionally reducing the threshold given that ragequit (or quitting) is going to happen regardless of the size of the fork. It is the forking group that could share the same goal and direction in a new DAO that matters.

[08] Prolonged process due to updatable state

The introduction of `updatePeriodEndBlock` in V3 compared to the V1/V2 could unnecessarily prolong the entire proposal voting process.

Consider reducing the pending period to make room for the updatable period which should nonetheless be entailing a longer period still, albeit in a more reasonable sense.

gzeon (judge) commented:

> **Reserve price not fully taken care of** -> Low

> **Code and comment mismatch** -> Non-Critical

> **Spelling errors** -> Non-Critical

> **Wrong adoption of block time** -> Non-Critical

> `MIN_PROPOSAL_THRESHOLD_BPS` **is too low a value** -> Low

> **V3 upgrade could depreciate token value due to oversupply** -> Non-Critical

> **Nouns fork may not efficiently remedy a bad situation** -> Non-Critical

> **Prolonged process due to updatable state** -> Non-Critical

Gas Optimizations

For this audit, 15 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **c3phas** received the top score from the judge.

The following wardens also submitted reports: JCN, Raihan, flutter_developer, petrichor, 0xAnah, naman1778, SM3_SS, SAQ, Aymen0909, MohammedRizwan, klau5, hunter_w3b, K42, *and* dharma09.

Notes from Warden

Warden's Disclaimer

While we try our best to maintain readability in the provided code snippets, some functions have been truncated to highlight the affected portions.

It's important to note that during the implementation of these suggested changes, developers must exercise caution to avoid introducing vulnerabilities. Although the optimizations have been tested prior to these recommendations, it is the responsibility of the developers to conduct thorough testing again.

Code reviews and additional testing are strongly advised to minimize any potential risks associated with the refactoring process.

Note on Gas estimates

I've tried to give the exact amount of gas being saved from running the included tests. Whenever the function is within the test coverage, the average gas before and after will be included, and often a diff of the code will also accompany this.

Some functions are not covered by the test cases or are internal/private functions. In this case, the gas can be estimated by looking at the opcodes involved. For some benchmarks are based on the function that calls this internal functions.

[G-01] Tightly pack storage variables/optimize the order of variable declaration

The EVM works with 32 byte words. Variables less than 32 bytes can be declared next to eachother in storage and this will pack the values together into a single 32 byte storage slot (if the values combined are <= 32 bytes). If the variables packed together are retrieved together in functions we will effectively save ~2000 gas with every subsequent SLOAD for that storage slot. This is due to us incurring a `Gwarmaccess (100 gas)` versus a `Gcoldsload (2100 gas)`. Here, the storage variables can be tightly packed from:

▶ Details

[G-02] Pack structs by putting data types that can fit together next to each other

As the solidity EVM works with 32 bytes, variables less than 32 bytes should be packed inside a struct so that they can be stored in the same slot, this saves gas when writing to storage ~20000 gas

We have some uint32 that can be be packed with an address (Save 1 SLOT: 2.1K gas)

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/NounsDAOInterfaces.sol#L653-L717

Pack lastMinuteWindowInBlocks,objectionPeriodDurationInBlocks,proposalUpdatablePeriodInBlocks with forkDAOTreasury

```
File: /packages/nouns-contracts/contracts/governance/NounsDAOInt
    struct StorageV3 {
                    <Truncated>
690:        uint32 lastMinuteWindowInBlocks;
```

```
        /// @notice Length of the objection period in blocks
        uint32 objectionPeriodDurationInBlocks;
        /// @notice Length of proposal updatable period in k
        uint32 proposalUpdatablePeriodInBlocks;
        /// @notice address of the DAO's fork escrow contrac
        INounsDAOForkEscrow forkEscrow;
        /// @notice address of the DAO's fork deployer contr
        IForkDAODeployer forkDAODeployer;
        /// @notice ERC20 tokens to include when sending fur
        address[] erc20TokensToIncludeInFork;
        /// @notice The treasury contract of the last deploy
        address forkDAOTreasury;
    <Truncated>

    }
```

```
diff --git a/packages/nouns-contracts/contracts/governance/Nouns
index 8fb0b4d3..ba0a251f 100644
--- a/packages/nouns-contracts/contracts/governance/NounsDAOInte
+++ b/packages/nouns-contracts/contracts/governance/NounsDAOInte
@@ -686,12 +686,6 @@ contract NounsDAOStorageV3 {
         // ================ V3 ================ //
         /// @notice user => sig => isCancelled: signatures that
         mapping(address => mapping(bytes32 => bool)) cancelledS
-        /// @notice The number of blocks before voting ends dur
-        uint32 lastMinuteWindowInBlocks;
-        /// @notice Length of the objection period in blocks
-        uint32 objectionPeriodDurationInBlocks;
-        /// @notice Length of proposal updatable period in bloc
-        uint32 proposalUpdatablePeriodInBlocks;
         /// @notice address of the DAO's fork escrow contract
         INounsDAOForkEscrow forkEscrow;
         /// @notice address of the DAO's fork deployer contract
@@ -700,6 +694,12 @@ contract NounsDAOStorageV3 {
         address[] erc20TokensToIncludeInFork;
         /// @notice The treasury contract of the last deployed
         address forkDAOTreasury;
+        /// @notice The number of blocks before voting ends dur
+        uint32 lastMinuteWindowInBlocks;
+        /// @notice Length of the objection period in blocks
+        uint32 objectionPeriodDurationInBlocks;
+        /// @notice Length of proposal updatable period in bloc
+        uint32 proposalUpdatablePeriodInBlocks;
         /// @notice The token contract of the last deployed for
```

```
        address forkDAOToken;
```

[G-03] Use calldata instead of memory for function parameters

If a reference type function parameter is read-only, it is cheaper in gas to use calldata instead of memory. Calldata is a non-modifiable, non-persistent area where function arguments are stored, and behaves mostly like memory.

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.

▶ Details

[G-04] Expensive operation inside a for loop

Function `quitInternal()` does a lot of inefficient operation mainly inside it’s for loops. I’ve optimized it as a whole but avoided some common optimizations

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/fork/newdao/governance/NounsDAOLogicV1Fork.sol#L218-L245

As it’s an internal one, the gas changes can be seen from the two function `quit`

`Benchmarks for quit(uint256[])`

	Min	Average	Median	Max
Before	743	245981	202066	501600
After	743	245388	201653	500096

`Benchmarks for quit(uint256[],address[])`

	Min	Average	Median	Max
Before	14309	165989	151098	347452
After	14309	165727	150891	346817

```
File: /packages/nouns-contracts/contracts/governance/fork/newdao
218:    function quitInternal(uint256[] calldata tokenIds, addre

223:        for (uint256 i = 0; i < tokenIds.length; i++) {
224:            nouns.transferFrom(msg.sender, address(timelock)
225:        }

231:        for (uint256 i = 0; i < erc20TokensToInclude.length;
232:            IERC20 erc20token = IERC20(erc20TokensToInclude[
233:            balancesToSend[i] = (erc20token.balanceOf(addres
234:        }

237:        timelock.sendETH(payable(msg.sender), ethToSend);
238:        for (uint256 i = 0; i < erc20TokensToInclude.length;
239:            if (balancesToSend[i] > 0) {
240:                timelock.sendERC20(msg.sender, erc20TokensTo
241:            }
242:        }
```

```
diff --git a/packages/nouns-contracts/contracts/governance/fork/
index 0b098e44..49d60515 100644
--- a/packages/nouns-contracts/contracts/governance/fork/newdao/
+++ b/packages/nouns-contracts/contracts/governance/fork/newdao/
@@ -219,25 +219,26 @@ contract NounsDAOLogicV1Fork is UUPSUpgrac
         checkGovernanceActive();

         uint256 totalSupply = adjustedTotalSupply();
-
+        NounsDAOExecutorV2  _timelock = timelock;
+        INounsTokenForkLike _nouns = nouns;
         for (uint256 i = 0; i < tokenIds.length; i++) {
-            nouns.transferFrom(msg.sender, address(timelock), t
+            _nouns.transferFrom(msg.sender, address(_timelock),
         }

         uint256[] memory balancesToSend = new uint256[](erc20To

         // Capture balances to send before actually sending the
```

```diff
-          uint256 ethToSend = (address(timelock).balance * tokenI
+          uint256 ethToSend = (address(_timelock).balance * toker
           for (uint256 i = 0; i < erc20TokensToInclude.length; i+
               IERC20 erc20token = IERC20(erc20TokensToInclude[i])
-              balancesToSend[i] = (erc20token.balanceOf(address(t
+              balancesToSend[i] = (erc20token.balanceOf(address(_
           }

           // Send ETH and ERC20 tokens
-          timelock.sendETH(payable(msg.sender), ethToSend);
+          _timelock.sendETH(payable(msg.sender), ethToSend);
           for (uint256 i = 0; i < erc20TokensToInclude.length; i+
               if (balancesToSend[i] > 0) {
-                  timelock.sendERC20(msg.sender, erc20TokensToInc
+                  _timelock.sendERC20(msg.sender, erc20TokensToIr
               }
           }
```

Don't read state inside loops, `escrow` and `forkId` should be cached outside the loop (Save 199 Gas on average)

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/fork/newdao/token/NounsTokenFork.sol#L148-L157

Gas benchmarks

	Min	Average	Median	Max
Before	6721	387833	217402	3683287
After	6754	387714	217439	3680633

```
File: /packages/nouns-contracts/contracts/governance/fork/newdac
149:        for (uint256 i = 0; i < tokenIds.length; i++) {
150:            uint256 nounId = tokenIds[i];
151:            if (escrow.ownerOfEscrowedToken(forkId, nounId)

156:            _mintWithOriginalSeed(msg.sender, nounId);
157:        }
```

```
    function claimFromEscrow(uint256[] calldata tokenIds) exter
+        INounsDAOForkEscrow _escrow =  escrow;
+        uint32 _forkId = forkId;
         for (uint256 i = 0; i < tokenIds.length; i++) {
             uint256 nounId = tokenIds[i];
-            if (escrow.ownerOfEscrowedToken(forkId, nounId) !=
+            if (_escrow.ownerOfEscrowedToken(_forkId, nounId) !

             _mintWithOriginalSeed(msg.sender, nounId);
         }
```

[G-05] Cache storage values in memory to minimize SLOADs

The code can be optimized by minimizing the number of SLOADs.

▶ Details

[G-06] Use the existing Local variable/global variable when equal to a state variable to avoid reading from state

Local variable `_escrow` should be used instead of reading `escrow`

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/fork/newdao/token/NounsTokenFork.sol#L131-L137

Gas benchmarks

	Min	Average	Median	Max
Before	211357	224893	211357	255180
After	211339	224875	211339	255162

```
File: /packages/nouns-contracts/contracts/governance/fork/newdac
131:        escrow = _escrow;

137:        NounsTokenFork originalToken = NounsTokenFork(addres
diff --git a/packages/nouns-contracts/contracts/governance/fork/
okenFork.sol
index a1f9d6d3..fea6fad8 100644
--- a/packages/nouns-contracts/contracts/governance/fork/newdao/
+++ b/packages/nouns-contracts/contracts/governance/fork/newdao/
@@ -134,7 +134,7 @@ contract NounsTokenFork is INounsTokenFork,
         remainingTokensToClaim = tokensToClaim;
         forkingPeriodEndTimestamp = _forkingPeriodEndTimestamp;

-        NounsTokenFork originalToken = NounsTokenFork(address(e
+        NounsTokenFork originalToken = NounsTokenFork(address(_
         descriptor = originalToken.descriptor();
         seeder = originalToken.seeder();
     }
```

Global variable `msg.sender` should be used instead of reading state (Save 128 gas on average)

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/NounsDAOLogicV2.sol#L240-L260

Gas benchmarks

	Min	Average	Median	Max
Before	13546	439476	380456	947962
After	13456	439348	380323	947829

```
File: /packages/nouns-contracts/contracts/governance/NounsDAOLog
240:        Proposal storage newProposal = _proposals[proposalCo

242:        newProposal.proposer = msg.sender;

260:        latestProposalIds[newProposal.proposer] = newPropos
```

We are setting `newProposal.proposer` to be equal to `msg.sender`. As `newProposal.proposer` is a state variable, it's a bit expensive to read, we can instead read `msg.sender` which is a global variable, thus more cheaper to read.

```diff
diff --git a/packages/nouns-contracts/contracts/governance/Nouns
index ff426a81..9328d9ce 100644
--- a/packages/nouns-contracts/contracts/governance/NounsDAOLogi
+++ b/packages/nouns-contracts/contracts/governance/NounsDAOLogi
@@ -257,7 +257,7 @@ contract NounsDAOLogicV2 is NounsDAOStorage\
         newProposal.totalSupply = temp.totalSupply;
         newProposal.creationBlock = block.number;

-        latestProposalIds[newProposal.proposer] = newProposal.i
+        latestProposalIds[msg.sender] = newProposal.id;
```

[G-07] Emitting storage values instead of the memory one

Here, the values emitted shouldn't be read from storage. The existing memory values should be used instead:

▶ Details

[G-08] Optimizing check order for cost efficient function execution

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.

▶ Details

[G-09] The following functions can benefit from some optimizations

▶ Details

[G-10] Nested if is cheaper than single statement

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/base/ERC721Checkpointable.sol#L243-L245

```
File: /packages/nouns-contracts/contracts/base/ERC721Checkpointa
243:         if (nCheckpoints > 0 && checkpoints[delegatee][nChec
244:             checkpoints[delegatee][nCheckpoints - 1].votes =
245         } else {          }
```

```
-        if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpo
-            checkpoints[delegatee][nCheckpoints - 1].votes = ne
+        if (nCheckpoints > 0) {
+            if( checkpoints[delegatee][nCheckpoints - 1].fromBl
+                checkpoints[delegatee][nCheckpoints - 1].votes
+            }
         } else {
```

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/fork/newdao/governance/NounsDAOLogicV1Fork.sol#L381-L383

```
File: /packages/nouns-contracts/contracts/governance/fork/newdao
381:         if (block.timestamp < delayedGovernanceExpirationTim
382:             revert WaitingForTokensToClaimOrExpiration();
383:         }
```

```
-        if (block.timestamp < delayedGovernanceExpirationTimest
-            revert WaitingForTokensToClaimOrExpiration();
+        if (block.timestamp < delayedGovernanceExpirationTimest
+            if ( nouns.remainingTokensToClaim() > 0) {
+                revert WaitingForTokensToClaimOrExpiration();
+            }
```

[G-11] Caching a variable that is used once just wastes Gas

No need to cache `ds.forkEscrow` as it's being used once

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/noun s-contracts/contracts/governance/fork/NounsDAOV3Fork.sol#L141-L160

```
File: /packages/nouns-contracts/contracts/governance/fork/NounsI
141:    function joinFork(
142:        NounsDAOStorageV3.StorageV3 storage ds,
143:        uint256[] calldata tokenIds,
144:        uint256[] calldata proposalIds,
145:        string calldata reason
146:    ) external {
147:        if (!isForkPeriodActive(ds)) revert ForkPeriodNotAct

149:        INounsDAOForkEscrow forkEscrow = ds.forkEscrow;
150:        address timelock = address(ds.timelock);
151:        sendProRataTreasury(ds, ds.forkDAOTreasury, tokenIds

153:        for (uint256 i = 0; i < tokenIds.length; i++) {
154:            ds.nouns.transferFrom(msg.sender, timelock, toke
155:        }

157:        NounsTokenFork(ds.forkDAOToken).claimDuringForkPeric

159:        emit JoinFork(forkEscrow.forkId() - 1, msg.sender, t
160:    }
```

```
diff --git a/packages/nouns-contracts/contracts/governance/fork/
index d87ffc70..4051da05 100644
--- a/packages/nouns-contracts/contracts/governance/fork/NounsDA
+++ b/packages/nouns-contracts/contracts/governance/fork/NounsDA
@@ -146,7 +146,6 @@ library NounsDAOV3Fork {
     ) external {
         if (!isForkPeriodActive(ds)) revert ForkPeriodNotActive

-        INounsDAOForkEscrow forkEscrow = ds.forkEscrow;
         address timelock = address(ds.timelock);
         sendProRataTreasury(ds, ds.forkDAOTreasury, tokenIds.le

@@ -156,7 +155,7 @@ library NounsDAOV3Fork {

         NounsTokenFork(ds.forkDAOToken).claimDuringForkPeriod(n

-        emit JoinFork(forkEscrow.forkId() - 1, msg.sender, toke
+        emit JoinFork(ds.forkEscrow.forkId() - 1, msg.sender, t
    }
```

NounsTokenFork.sol.claimDuringForkPeriod(): `_currentNounId` should not be cached

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/fork/newdao/token/NounsTokenFork.sol#L166-L185

The variable `currentNounId` is being once, as such no need to cache.

```
File: /packages/nouns-contracts/contracts/governance/fork/newdac
166:    function claimDuringForkPeriod(address to, uint256[] cal
167:        uint256 currentNounId = _currentNounId;

184:        if (maxNounId >= currentNounId) _currentNounId = max
```

[G-12] Importing an entire library while only using one function isn't necessary

https://github.com/nounsDAO/nouns-monorepo/blob/718211e063d511eeda1084710f6a682955e80dcb/packages/nouns-contracts/contracts/governance/NounsDAOV3Votes.sol#L22

```
File: /packages/nouns-contracts/contracts/governance/NounsDAOV3\
22:import { SafeCast } from '@openzeppelin/contracts/utils/math/

252:            proposal.objectionPeriodEndBlock = SafeCast.toUi
253:                proposal.endBlock + ds.objectionPeriodDurati
254:            );
```

We import the entire library `SafeCast` yet we only need to utilize one function from it ie `toUint64()` . Peeking into it's implementation from Openzeppelin we have the following https://github.com/OpenZeppelin/openzeppelin-

contracts/blob/7ccea54dc15856d0e6c3b61b829b85d9e52195cd/contracts/utils/math/SafeCast.sol#L441-L446.

```
File: /contracts/utils/math/SafeCast.sol
441:    function toUint64(uint256 value) internal pure returns (
442:        if (value > type(uint64).max) {
443:            revert SafeCastOverflowedUintDowncast(64, value)
444:        }
445:        return uint64(value);
446:    }
```

```
diff --git a/packages/nouns-contracts/contracts/governance/Nouns
index 3743132b..782a2cf2 100644
--- a/packages/nouns-contracts/contracts/governance/NounsDAOV3Vo
+++ b/packages/nouns-contracts/contracts/governance/NounsDAOV3Vo
@@ -19,13 +19,14 @@ pragma solidity ^0.8.19;

 import './NounsDAOInterfaces.sol';
 import { NounsDAOV3Proposals } from './NounsDAOV3Proposals.sol'
-import { SafeCast } from '@openzeppelin/contracts/utils/math/Sa

 library NounsDAOV3Votes {
     using NounsDAOV3Proposals for NounsDAOStorageV3.StorageV3;

     error CanOnlyVoteAgainstDuringObjectionPeriod();

+    error SafeCastOverflowedUintDowncast();
+
     /// @notice An event emitted when a vote has been cast on a
     /// @param voter The address which casted a vote
     /// @param proposalId The proposal id which was voted on
@@ -249,9 +250,11 @@ library NounsDAOV3Votes {
                 // second part of the vote flip check
                 !ds.isDefeated(proposal)
             ) {
-                proposal.objectionPeriodEndBlock = SafeCast.toUint6
-                    proposal.endBlock + ds.objectionPeriodDurationI
-                );
+
+                if (proposal.endBlock + ds.objectionPeriodDurationI
+                        revert SafeCastOverflowedUintDowncast()
+                }
+                proposal.objectionPeriodEndBlock = uint64(proposal.
                 emit ProposalObjectionPeriodSet(proposal.id, propos
             }
```

Alternatively we can implement our own internal function to do the safeCast.

Conclusion

It is important to emphasize that the provided recommendations aim to enhance the efficiency of the code without compromising its readability. We understand the value of maintainable and easily understandable code to both developers and auditors.

As you proceed with implementing the suggested optimizations, please exercise caution and be diligent in conducting thorough testing. It is crucial to ensure that the changes are not introducing any new vulnerabilities and that the desired performance improvements are achieved. Review code changes, and perform thorough testing to validate the effectiveness and security of the refactored code.

Should you have any questions or need further assistance, please don't hesitate to reach out.

Audit Analysis

For this audit, 5 analysis reports were submitted by wardens. An analysis report examines the codebase as a whole, providing observations and advice on such topics as architecture, mechanism, or approach. The report highlighted below by **0xnev** received the top score from the judge.

The following wardens also submitted reports: 0xSmartContract, shark, Matin, K42, and ihtishamsudo.

[01] Summary of Codebase

1.1 Description

Nouns is a generative NFT project on Ethereum, where a new Noun is minted and auctioned off every day, and each token represents one vote where proposers who hold a noun and create and vote on governance proposals, which execute transactions on the ethereum blockchain when approved.

1.2 Proposal States and flow

To summarize the NounsDAO protocol, it will be helpful to look at the various states introduced in the NounsDaoV3 contracts. But first lets look at the creation of proposals.

1.2.1 Creation of Proposal

- Propose the proposal via `propose/proposeOnTimelockV1/proposeBySigs`
- Check that proposer (and if there is signers), have sufficient votes to meet minimum voting threshold
- Check validity of signatures if there are signers
- Check transactions validity (array length check and maximum actions (10) allowed check)
- Check that there is no active proposal for proposer (NounsDao only allows 1 active proposal per proposer)
- Create a new proposal if check passes, at this stage, proposal will be in the Updatable state

1.2.2 Updatable

- At this stage, proposals are not active for voting yet, and there exists a updatable period for proposers/signers to edit proposals transaction details and description via `updateProposal()/updateProposalTransactions()/updateProposalBySigs()` and `updateProposalDescription()` respectively

1.2.3 Pending

- Once updatable period ends, the proposal reaches the pending state, where it switches to Active once the `block.number` reaches the starting block.

1.2.4 Active

- During the Active state, voters can start casting votes, where the uint8 `support` represents the vote value, with 0 = against, 1 = for and 2 = abstain

- Vote casting can be done via `castVote()/castVoteWithReason()/castVoteBySig()`
- Voting casting can also be done with a request for gas refund from DAO via `castRefundableVote()/castRefundableVoteWithReason()`
- In the Active state, there exists 3 state transitions:

  1. During the active period if there are enough votes that exceeds quorum, and for votes more than against votes, then proposal state will switch to Succeeded.
  2. If the opposite occurs where against votes are more than for votes, proposal state will switch to Defeated
  3. The NounsDaoV3 introduced a new mechanism known as the objection period, where against voters are given more reaction time to react to last minute state transitions from Defeated to Succeeded. A more detailed explanation is included in 1.2.5

1.2.5 ObjectionPeriod

- The ObjectionPeriod is a conditional state where any last minute votes swinging a already Defeated proposal to Succeeded by for voters supporting proposal will trigger the state
- In this period, only against voters are allowed to vote to allow them to swing the proposal back to the Defeated state.
- If enough against voters votes against proposal, then the final proposal state will be Defeated, and proposal will not be queued and executed

1.2.6 Queued and Executed

- If the current state of proposal after Active period of voting is Succeeded, then proposal is ready to be queued and executed by admin via the `NounsDaoExecutorV2.sol` contract
- More specifically, proposals are queued using `queueTransaction()` and executed via `executeTransaction()`

1.2.7 Expired

- If proposals are not executed within 21 days (increased from 14 days to account for possible forking period) after being queued, then it will expire

1.2.8 Cancelled

- At any point of time, proposers and/or signers can cancel active proposals as long as proposal has not reached a final state, specifically following states (`Canceled/Defeated/Expired/Executed/Vetoed`).

1.2.9 Vetoed

- The Vetoed state essentially means that proposal is Cancelled by vetoer set by NounsDAO
- It is a means for NounsDAO to protect the protocol against malicious proposals.

1.3 Forking Mechanism

- There is also a new forking mechanism that allows forking of a new Noun Dao if enough Noun tokens are escrowed
- This is wonderfully summarized by the protocol here

[02] Architecture Improvements

2.1 Consider allowing for voters to recover from swing last minute swing from successful to defeated

Currently, the Nouns Dao introduce a objection period where there is a last-minute proposal swing from defeated to successful, affording against voters more reaction time.

This could be unfair to the for voters, where the reverse could happen, when there is a last-minute proposal swing from successful to defeated, but no time is allowed for for voters to react. Hence, protocol could introduce a new mechanism/state to allow this to happen, where similarly, only a against vote casted in the last minute voting block can trigger this period.

2.2 Consider implementing a mechanism to unvote and/or updated vote choice

Currently, there are no mechanisms for voters to unvote or update their votes, and they can only ever vote once due to the `hasVoted` flag. Consider implementing a mechanism to unvote and/or update vote choice.

2.3 Consider not allowing creation of new proposal when current proposal is still in Queued state

Queued proposals are still active, since it has not been executed. As such, consider not allowing creation of proposal when proposal state is queued. Given Nouns Dao only allow 1 active proposal per proposer, there could be a scenario where there are multiple proposals queued if proposals are not yet executed.

2.4 Consider not allowing executing fork if not a token holder

In the contest Docs, it is stated that any token holder can execute fork if fork threshold is met. However, anybody, not just token holders can execute fork via `executeFork()` once threshold is met. Since Nouns govern Noun DAO, consider only allowing only token holders to execute fork.

[03] Centralization risks

3.1 Vote casters may lose gas refund if contract is underfunded `_refundGas`

Any one can fund the `NounsDAOV3Votes.sol` contract, but it is presumably the DAO funding it to refund gas for voters. If contract ETH balance is insufficient, voters may not get refunded their gas when voting.

3.2 DAO can affect proposal thresholds anytime by adjusting totalSupply

In the new NounsDaoV3Logic, all proposal thresholds are calculated using an adjusted total supply instead of the fixed nouns supply previously. This adjusted total supply represents the total supply of nouns minus nouns owed by DAO. If in any of the address the Nouns DAO mints/transfer nouns tokens, it could affect proposal thresholds by increasing/decreasing it respectfully, potentially preventing/allowing proposals to be created.

3.3 DAO can close escrow and withdraw escrowed tokens anytime

In `NounsDAOForkEscrow.sol` , any nouns tokens sent to the contract to be escrowed faces a potential risk of DAO closing escrow at anytime, essentially locking up the tokens and cannot be unescrowed, with the tokens only being able to be withdrawn by the DAO.

3.4 `NounsDAOV3Proposals.cancel()` : Signers can collude to cancel proposer proposal anytime by adjusting voting power

With the introduction of proposing proposals with other signers, it also gives the power to signers to cancel proposals at anytime without the need to consult proposer/other signers. This opens up the ability for any signers or even the proposer to invalidate votes simply by cancelling the proposal if they do not agree with the direction of the state that proposal is approaching (`Defeated/Succeeded`).

[04] Time Spent

- Day 1: Compare v2 and v3 NounsDao versions, noting new mechanisms such as proposal editing, proposal by sig and objection only period.
- Day 2: Audit NounsV3Logic coupled with NounsV3Proposals
- Day 3: Audit NounsV3Logic coupled with NounsV3Vote and NoundsDAOV3Admin
- Day 4: Finish up Analysis

Time spent:

48 hours

Disclosures

Top