Arbitrum Security Council Election System Findings & Analysis Report

2023-09-20

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 Arbitrum Security Council Election System smart contract system written in Solidity. The audit took place between August 3—August 10 2023.

Wardens

38 Wardens contributed reports to the Arbitrum Security Council Election System audit:

1. Daniel526
2. MiloTruck

3. dirk_y
4. HE1M
5. KingNFT
6. AkshaySrivastav
7. hals
8. ktg
9. 0xTheC0der
10. Kow
11. Sathish9098
12. catellatech
13. K42
14. 0xSmartContract
15. LeoS
16. berlin-101
17. 0xAnah
18. JCK
19. 0xnev
20. MSK
21. yixxas
22. kodyvim
23. Udsen
24. rektthecode (Angry_Mustache_Man, devblixt, and VictoryGod)
25. petrichor
26. naman1778
27. SAAJ
28. dharma09
29. caventa
30. ernestognw
31. eierina

32. Oxprinc
33. Oxbepresent
34. arialblack14
35. Mirror
36. nobody2018

This audit was judged by Oxean.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 6 unique vulnerabilities. Of these vulnerabilities, 1 received a risk rating in the category of HIGH severity and 5 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 13 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 Arbitrum Security Council Election System repository, and is composed of 20 smart contracts written in the Solidity programming language and includes 2,184 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] Signatures can be replayed in `castVoteWithReasonAndParamsBySig()` to use up more votes than a user intended

Submitted by MiloTruck, *also found by* KingNFT *and* HE1M

In the `SecurityCouncilNomineeElectionGovernor` and `SecurityCouncilMemberElectionGovernor` contracts, users can provide a signature to allow someone else to vote on their behalf using the `castVoteWithReasonAndParamsBySig()` function, which is in Openzeppelin's `GovernorUpgradeable`:

GovernorUpgradeable.sol#L480-L495

```
address voter = ECDSAUpgradeable.recover(
    _hashTypedDataV4(
        keccak256(
            abi.encode(
                EXTENDED_BALLOT_TYPEHASH,
                proposalId,
                support,
                keccak256(bytes(reason)),
                keccak256(params)
            )
        )
    ),
    v,
    r,
    s
);
```

As seen from above, the signature provided does not include a nonce. This becomes an issue in nominee and member elections, as users can choose not to use all of their votes in a single call, allowing them split their voting power amongst contenders/nominees:

[Nominee Election Specification]

> A single delegate can split their vote across multiple candidates.

[Member Election Specification]

> Additionally, delegates can cast votes for more than one nominee:

- Split voting. delegates can split their tokens across multiple nominees, with 1 token representing 1 vote.

Due to the lack of a nonce, `castVoteWithReasonAndParamsBySig()` can be called multiple times with the same signature.

Therefore, if a user provides a signature to use a portion of his votes, an attacker can repeatedly call `castVoteWithReasonAndParamsBySig()` with the same signature to use up more votes than the user originally intended.

Impact

Due to the lack of signature replay protection in `castVoteWithReasonAndParamsBySig()`, during nominee or member elections, an attacker can force a voter to use more votes on a contender/nominee than intended by replaying his signature multiple times.

Proof of Concept

Assume that a nominee election is currently ongoing:

- Bob has 1000 votes, he wants to split his votes between contender A and B:
    - He signs one signature to give 500 votes to contender A.
    - He signs a second signature to allocate 500 votes to contender B.
- `castVoteWithReasonAndParamsBySig()` is called to submit Bob's first signature:

    - This gives contender A 500 votes.
- After the transaction is executed, Alice sees Bob's signature in the transaction.
- As Alice wants contender A to be elected, she calls `castVoteWithReasonAndParamsBySig()` with Bob's first signature again:
    - Due to a lack of a nonce, the transaction is executed successfully, giving contender A another 500 votes.
- Now, when `castVoteWithReasonAndParamsBySig()` is called with Bob's second signature, it reverts as all his 1000 votes are already allocated to contender A.

In the scenario above, Alice has managed to allocate all of Bob's votes to contender A against his will. Note that this can also occur in member elections, where split voting is also allowed.

Recommended Mitigation

Consider adding some form of signature replay protection in the `SecurityCouncilNomineeElectionGovernor` and `SecurityCouncilMemberElectionGovernor` contracts.

One way of achieving this is to override the `castVoteWithReasonAndParamsBySig()` function to include a nonce in the signature, which would protect against signature replay.

DZGoldman (Arbitrum) confirmed and commented via duplicate issue `#173`:

> Confirmed; see here for fix.

Medium Risk Findings (5)

[M-01] `SecurityCouncilMemberElectionGovernor` Owner Can Change `votingPeriod` During an Active Election

Submitted by hals, also found by MiloTruck, HE1M, and Kow

https://github.com/ArbitrumFoundation/governance/blob/c18de53820c505fc459f766c1b224810eaeaabc5/src/security-council-mgmt/governors/SecurityCouncilMemberElectionGovernor.sol#L103-L110
https://github.com/ArbitrumFoundation/governance/blob/c18de53820c505fc459f766c1b224810eaeaabc5/src/security-council-mgmt/governors/modules/SecurityCouncilMemberElectionGovernorCountingUpgradeable.sol#L77-L84

Impact

- In `SecurityCouncilMemberElectionGovernor` contract : `relay` function enables the contract owner from making calls to any contract address.
- And in `SecurityCouncilMemberElectionGovernorCountingUpgradeable` contract: `setFullWeightDuration` can be accessed only by invoking it from `SecurityCouncilMemberElectionGovernor` which is possible only via `relay` function.
- So the owner can set a new value for `fullWeightDuration` that is used to determine the deadline after which the voting weight will linearly decrease.
- But when setting it; there's no check if there's a current active proposal.
- This makes the voting unfair and the results unreliable as the owner can control the voting power during the election; as increasing the voting power of late voters if `fullWeightDuration` is set to a higher value during active election.

Proof of Concept

- Code:

  SecurityCouncilMemberElectionGovernor contract/relay function

```
File:governance/src/security-council-mgmt/governors/SecurityCouncilMemb
Line 103-110:
    function relay(address target, uint256 value, bytes calldata data)
        external
        virtual
        override
        onlyOwner
    {
        AddressUpgradeable.functionCallWithValue(target, data, value);
    }
```

[SecurityCouncilMemberElectionGovernorCountingUpgradeable contract/setFullWeightDuration function]

```
File: governance/src/security-council-mgmt/governors/modules/SecurityCo
Line 77-84:
   function setFullWeightDuration(uint256 newFullWeightDuration) public
      if (newFullWeightDuration > votingPeriod()) {
          revert FullWeightDurationGreaterThanVotingPeriod(newFullWeigh
      }

      fullWeightDuration = newFullWeightDuration;
      emit FullWeightDurationSet(newFullWeightDuration);
  }
```

- Foundry PoC:
- `testSetVotingPeriodDuringActiveProposal()` test is added to `SecurityCouncilMemberElectionGovernorTest.t.sol` file; where the relay function is invoked by the contract owner to change the `fullWeightDuration` during an active proposal:

```
function testSetVotingPeriodDuringActiveProposal() public {
 //1. initiate a proposal
 _propose(0);
 //2. change fullWeightDuration while the proposal is still active
 assertEq(governor.votingPeriod(), initParams.votingPeriod);
 vm.prank(initParams.owner);
 governor.relay(
   address(governor),
   0,
   abi.encodeWithSelector(governor.setVotingPeriod.selector, 121_212)
 );
 assertEq(governor.votingPeriod(), 121_212);
}
```

- Test result:

```
$ forge test --match-test testSetVotingPeriodDuringActiveProposal
[PASS] testSetVotingPeriodDuringActiveProposal() (gas: 118129)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 4.51ms
Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

Tools Used

Manual Testing & Foundry.

Recommended Mitigation Steps

Enable setting a new value for `fullWeightDuration` only if there's no active election.

Assessed type

Governance

yahgwai (Arbitrum) confirmed and commented via duplicate issue `#52`:

> Fix: https://github.com/ArbitrumFoundation/governance/pull/184. Fix is just add comments informing the caller not to set during certain times.

[M-02] `SecurityCouncilNomineeElectionGovernor` might have to wait for more than 6 months to create election again

Submitted by **ktg**, *also found by* **MiloTruck**, **KingNFT**, *and* **0xTheC0der**

According to the document (https://forum.arbitrum.foundation/t/proposal-security-council-elections-proposed-implementation-spec/15425/1#h-1-nominee-selection-7-days-10), security council election can be created every 6 months. Contract `SecurityCouncilNomineeElectionGovernor` implements this by these codes:

```
    function createElection() external returns (uint256 proposalId)
        // require that the last member election has executed
        _requireLastMemberElectionHasExecuted();

        // each election has a deterministic start time
        uint256 thisElectionStartTs = electionToTimestamp(electi
        if (block.timestamp < thisElectionStartTs) {
            revert CreateTooEarly(block.timestamp, thisElectionS
        }
        ...
}

    function electionToTimestamp(uint256 electionIndex) public v
        // subtract one to make month 0 indexed
```

```
        uint256 month = firstNominationStartDate.month - 1;

        month += 6 * electionIndex;
        uint256 year = firstNominationStartDate.year + month / 1
        month = month % 12;

        // add one to make month 1 indexed
        month += 1;

        return DateTimeLib.dateTimeToTimestamp({
            year: year,
            month: month,
            day: firstNominationStartDate.day,
            hour: firstNominationStartDate.hour,
            minute: 0,
            second: 0
        });
    }
```

If `electionIndex` = 1, function `createElection` will call `electionToTimestamp` to calculate for timestamp 6 months from `firstNominationStartDate`. However, the code uses `firstNominationStartDate.day` to form the result day:

```
        return DateTimeLib.dateTimeToTimestamp({
            year: year,
            month: month,
            day: firstNominationStartDate.day,
            hour: firstNominationStartDate.hour,
            minute: 0,
            second: 0
        });
```

This could result in wrong calculation because the day in months can varies from 28-31. Therefore, the worst case is that the user has to wait for 6 months + 4 more days to create new election.

For example, if `firstNominationStartDate` = `2024-08-31-01:00:00` (which is the last day of August). The user might expect that they can create election again 6 months from that, which mean 1:00 AM of the last day of February 2025 (which is

`2025-02-28-01:00:00`), but in fact the result of `electionToTimestamp` would be `2025-03-03-01:00:00`, 4 days from that.

Below is POC for the above example, for easy of testing, place this test case in file `test/security-council-mgmt/governors/SecurityCouncilNomineeElectionGovernor.t.sol` under contract `SecurityCouncilNomineeElectionGovernorTest` and run it using command:

`forge test --match-path test/security-council-mgmt/governors/SecurityCouncilNomineeElectionGovernor.t.sol --match-test testDateTime -vvvv`

```
        function testDateTime() public {
            // Deploy a new governor
            // with first nomination start date = 2024-08-30T01:00
            SecurityCouncilNomineeElectionGovernor  newGovernor = _d

            SecurityCouncilNomineeElectionGovernor.InitParams memory
            =       SecurityCouncilNomineeElectionGovernor.InitParam
                firstNominationStartDate: Date({year: 2024, month: 8
                nomineeVettingDuration: 1 days,
                nomineeVetter: address(0x11),
                securityCouncilManager: ISecurityCouncilManager(addr
                securityCouncilMemberElectionGovernor: ISecurityCoun
                    payable(address(0x33))
                ),
                token: IVotesUpgradeable(address(0x44)),
                owner: address(0x55),
                quorumNumeratorValue: 20,
                votingPeriod: 1 days
            });

            // The next selection is not available until timestamp 1
            // which is 2025-03-03T1:00:00 AM GMT
            newGovernor.initialize(newInitParams);
            assertEq(newGovernor.electionToTimestamp(1), 1740963600)

        }
```

You can use an online tool like https://www.epochconverter.com/ to check that `1740963600` is `Monday, March 3, 2025 1:00:00 AM GMT`.

Recommended Mitigation Steps

I recommend fixing the math so that the duration between elections are exactly 6 months like documented.

Assessed type

Math

yahgwai (Arbitrum) confirmed and commented:

> Fix: https://github.com/ArbitrumFoundation/governance/pull/180 - added additional checks to deploy script.

[M-03] Incorrect initialization of `SecurityCouncilMemberRemovalGovernor` contract

Submitted by AkshaySrivastav, also found by MiloTruck

The `SecurityCouncilMemberRemovalGovernor` contract inherits `GovernorUpgradeable` & `EIP712Upgradeable` contracts but does not invoke their individual initialzers during its own initialization. Due to which the state of `GovernorUpgradeable` & `EIP712Upgradeable` contracts remain uninitialized.

```
contract SecurityCouncilMemberRemovalGovernor is
    GovernorUpgradeable,
    GovernorVotesUpgradeable,
    ...
{
    function initialize(
        ...
    ) public initializer {
        __GovernorSettings_init(_votingDelay, _votingPeriod, _pr
        __GovernorCountingSimple_init();
        __GovernorVotes_init(_token);
        __ArbitrumGovernorVotesQuorumFraction_init(_quorumNumera
        __GovernorPreventLateQuorum_init(_minPeriodAfterQuorum);
        __ArbitrumGovernorProposalExpirationUpgradeable_init(_pr
        _transferOwnership(_owner);
        ...
```

```
        }
    }


    abstract contract GovernorUpgradeable is EIP712Upgradeable, ...
        function __Governor_init(string memory name_) internal onlyI
            __EIP712_init_unchained(name_, version());
            __Governor_init_unchained(name_);
        }
        function __Governor_init_unchained(string memory name_) inte
            _name = name_;
        }
    }
```

Due to this issue:

- In `GovernorUpgradeable` the `_name` storage variable is never initialized and the `name()` function returns an empty string.
- In `EIP712Upgradeable` the `_HASHED_NAME` storage variable is never initialized. This variable is used in the `castVoteBySig` & `castVoteWithReasonAndParamsBySig` functions of `SecurityCouncilMemberRemovalGovernor`.

The non-initialization of `EIP712Upgradeable` contract breaks the compatibility of `SecurityCouncilMemberRemovalGovernor` contract with the EIP712 standard.

It should be noted that the other contracts like `SecurityCouncilMemberElectionGovernor` and `SecurityCouncilNomineeElectionGovernor` also inherits the `GovernorUpgradeable` & `EIP712Upgradeable` contracts and initializes them correctly. So the incorrect initialization of `SecurityCouncilMemberRemovalGovernor` also breaks the overall code consistency of protocol.

The `SecurityCouncilMemberElectionGovernor` and `SecurityCouncilNomineeElectionGovernor` contract also force users to do voting only via `castVoteWithReasonAndParams` and `castVoteWithReasonAndParamsBySig` functions. Hence it can be clearly observed

that the protocol wants to utilize voting-by-signature feature, which requires EIP712 compatibility.

Proof of Concept

This test case was added in `test/security-council-mgmt/SecurityCouncilMemberRemovalGovernor.t.sol` and ran using `forge test --mt test_audit.`

```
function test_audit_notInvoking___Governor_init() public {
    assertEq(scRemovalGov.name(), "");
    assertEq(bytes(scRemovalGov.name()).length, 0);
}
```

Tools Used

Foundry

Recommended Mitigation Steps

Consider initializing the `GovernorUpgradeable` & `EIP712Upgradeable` contracts in `SecurityCouncilMemberRemovalGovernor.initialize` function.

```
function initialize(
    ...
) public initializer {
    __Governor_init("SecurityCouncilMemberRemovalGovernor");
    ...
}
```

sorrynotsorry (lookout) commented:

> Out of scope —> Already included in automated findings report: [L-03] Upgradeable contract not initialized

AkshaySrivastav (warden) commented:

> @0xean - I would like to add more context for this report.

> I was aware that this issue has been included in the automated findings but still reported this bug due to following reasons:

- The automated report does not explain the full impact of the finding and hence incorrectly judges it as low.
- Almost all cases mentioned in `Upgradeable contract not initialized` section are false positives.

> False positives reported in automated report:

- `SecurityCouncilManager` does not need to invoke `__AccessControl_init` as `__AccessControl_init` is an empty function.
- `SecurityCouncilMemberElectionGovernor` does not need to invoke `__Ownable_init` because the contract wants to provide ownership to a input parameter `_owner` while the `__Ownable_init` gives ownership to `msg.sender`. To achieve that the `SecurityCouncilMemberElectionGovernor` invokes the `_transferOwnership` function with `_owner` parameter in the initializer.
- > `SecurityCouncilMemberRemovalGovernor`:
  - invokes `__ArbitrumGovernorProposalExpiration_init` correctly.
  - handles the `__Ownable_init` case correctly similar to above explained `SecurityCouncilMemberElectionGovernor` case.
  - does not invoke `__Governor_init`, this is the issue reported by me.
- `SecurityCouncilNomineeElectionGovernor` handles the `__Ownable_init` case correctly similar to above explained `SecurityCouncilMemberElectionGovernor` case.

> Due to the above explained reasons I separately reported this bug as `medium` severity. The bug leads to incompatibility of a protocol contract with EIP712 standard which the contract always wanted to adhere to.

> Hope this report can be reviewed again. Thanks.

MiloTruck (warden) commented:

> @0xean - I would like to add on to what was stated above as my issue #253 is a duplicate.

> The C4 docs explicitly states that raising issues from bot reports to a higher severity is fair game, as seen here:
>
> Wardens may use automated tools as a first pass, and build on these findings to identify High and Medium severity issues ("HM issues"). However, submissions based on automated tools will have a higher burden of proof for demonstrating to sponsors a relevant HM exploit path in order to be considered satisfactory.
>
> In my issue, I've demonstrated how the missing `__Governor_init()` call results in incorrect signature verification in `castVoteBySig()` and `castVoteWithReasonAndParamsBySig()` and violates the EIP-712 standard, which in my opinion is deserving of medium severity.
>
> Additionally, as pointed out above, it would have been extremely easy for a sponsor to miss this medium finding amongst all the other false positives in the bot report.
>
> As such, I don't think this finding should be ruled out as OOS as we've demonstrated how it has a greater severity than stated in the bot report and provided additional value to the sponsor.
>
> Would like to hear your opinion as a judge, thanks!

0xean (judge) commented:

> Thanks for the comments. After reviewing, I think raising this to Medium and awarding this is fair. The bot report is vague on impact and without this being called to the sponsor's attention explicitly with a much broader impact, it could go unfixed until a later date.
>
> Will welcome sponsor comments before making a final call.

yahgwai (Arbitrum) commented:

> Issue was well written and had correct mitigation. I think Medium seems fair. Thank you for finding it and pointing it out!
>
> Fix here:
> https://github.com/ArbitrumFoundation/governance/commit/1a6b66d23f1226e93f0d67c3ed8e9fe1174c178c

[M-04] Security Council can undermine any DAO votes to remove a member

Submitted by dirk_y

https://github.com/ArbitrumFoundation/governance/blob/c18de53820c505fc459f766c1b224810eaeaabc5/src/security-council-mgmt/SecurityCouncilManager.sol#L183-L190
https://github.com/ArbitrumFoundation/governance/blob/c18de53820c505fc459f766c1b224810eaeaabc5/src/security-council-mgmt/SecurityCouncilManager.sol#L161-L173
https://github.com/ArbitrumFoundation/governance/blob/c18de53820c505fc459f766c1b224810eaeaabc5/src/security-council-mgmt/SecurityCouncilManager.sol#L176-L180
https://github.com/ArbitrumFoundation/governance/blob/c18de53820c505fc459f766c1b224810eaeaabc5/src/security-council-mgmt/SecurityCouncilManager.sol#L143-L159

As stated in the updated constitution text:

"Security Council members may only be removed prior to the end of their terms under two conditions:

1. At least 10% of all Votable Tokens have casted votes "in favour" of removal and at least 5/6 (83.33%) of all casted votes are "in favour" of removal; or
2. At least 9 of the Security Council members vote in favour of removal."

However, since the Security Council are the only party with the ability to add a member, they can simply re-add the member that was removed by a DAO vote if they disagree with the vote. This entirely defeats the point of giving the DAO the ability to remove a member of the council mid-term.

Proof of Concept

A member of the Security Council can be removed mid-term by calling `removeMember` in `SecurityCouncilManager.sol`. This can only be called by addresses that have the `MEMBER_REMOVER_ROLE` role; the `SecurityCouncilMemberRemovalGovernor.sol` contract and the 9 of 12 emergency Security Council:

```
function removeMember(address _member) external onlyRole(MEM
    if (_member == address(0)) {
        revert ZeroAddress();
    }
    Cohort cohort = _removeMemberFromCohortArray(_member);
    _scheduleUpdate();
    emit MemberRemoved({member: _member, cohort: cohort});
}
```

The actual member removal is performed by `_removeMemberFromCohortArray`:

```
function _removeMemberFromCohortArray(address _member) inter
    for (uint256 i = 0; i < 2; i++) {
        address[] storage cohort = i == 0 ? firstCohort : se
        for (uint256 j = 0; j < cohort.length; j++) {
            if (_member == cohort[j]) {
                cohort[j] = cohort[cohort.length - 1];
                cohort.pop();
                return i == 0 ? Cohort.FIRST : Cohort.SECOND
            }
        }
    }
    revert NotAMember({member: _member});
}
```

As you can see, the member that has been removed isn't tracked/stored anywhere for later querying. So, let's assume that the DAO has successfully voted to remove a member. The `removeMember` method is called when the vote is successful and the update is schedule in the L2 timelock.

The Security Council now have the ability (as the only party with the `MEMBER_ADDER_ROLE` role) to add a new member to fill the relevant cohort on the council by calling `addMember`:

```
function addMember(address _newMember, Cohort _cohort) exter
    _addMemberToCohortArray(_newMember, _cohort);
    _scheduleUpdate();
    emit MemberAdded(_newMember, _cohort);
}
```

Where `_addMemberToCohortArray` looks like:

```
function _addMemberToCohortArray(address _newMember, Cohort
    if (_newMember == address(0)) {
        revert ZeroAddress();
    }
    address[] storage cohort = _cohort == Cohort.FIRST ? fir
    if (cohort.length == cohortSize) {
        revert CohortFull({cohort: _cohort});
    }
    if (firstCohortIncludes(_newMember)) {
        revert MemberInCohort({member: _newMember, cohort: C
    }
    if (secondCohortIncludes(_newMember)) {
        revert MemberInCohort({member: _newMember, cohort: C
    }

    cohort.push(_newMember);
}
```

As you can see, this method only checks to ensure the member being added doesn't already exist in either cohort, however there is no validation whether or not this member has been removed before in the current term. As a result, any DAO votes to remove a member can be usurped by the Security Council since they can simply restate the member that was voted to be removed.

Recommended Mitigation Steps

Any member removals mid-term should be tracked for the lifecycle of the term (6 months) to ensure that the same member can't be re-added. This array should be wiped when a cohort is replaced with a call to `replaceCohort`.

Assessed type

Invalid Validation

yahgwai (Arbitrum) acknowledged and commented:

> Acknowledge: The system is working as intended here. The constitution does not specify who the council can or cannot add. This issue is therefore a critique of the

> constitution (which was not in scope) rather than the code. However we appreciate this critique of the constitution and believe it to be a valid concern.

dirk_y (warden) commented:

> Just wanted to add some more context here as I'm expecting some debate around this.
>
> In my view the Security Council could still be acting in good faith but simply disagree with the vote of the DAO. The security council might have some inside knowledge or existing relationships with the removed member, and they still trust the removed member and trust them to act in good faith for any emergency/non-emergency votes. Maybe they're simply the best person for the job.
>
> Imagine a democratic vote for a new prime minister where the existing prime minister simply declines the result of the vote and remains in office anyway...i.e. a dictatorship. The dictator still believes they're acting in the best interests of the nation state. In summary the security council can still be acting in good faith, but their actions are undermining trust in the system (in particular the DAO is losing trust in the security council, despite the security council believing they are acting in the best interests of the DAO).
>
> And yes you could argue this is a critique of the constitution but I actually believe this is a critique of the current implementation. Similarly to using OZ roles, you could have a role that can undermine another role's action; this could be a valid medium assuming the two roles weren't owned by the same user.

[M-05] Inconsistent Cohort Replacement Process in Security Council Manager

Submitted by Daniel526*, also found by* Daniel526

The contract `SecurityCouncilManager` in the Arbitrum DAO's security council system has an inconsistency in its cohort replacement process. This inconsistency could potentially lead to a situation where a member is present in both the old and new cohorts simultaneously, contradicting the governance rules set out in the DAO's constitution.

Impact

The inconsistency in the cohort replacement process could lead to a situation where a member is elected to a new cohort while also remaining in the old cohort during ongoing elections. This can create confusion, undermine the democratic process, and potentially compromise the integrity of Security Council decisions. Such a scenario directly contradicts the DAO's constitution, which requires careful handling of cohort replacement to avoid conflicts.

Proof of Concept

The Arbitrum DAO's constitution outlines a careful process for replacing cohorts of Security Council members to avoid race conditions and operational conflicts, particularly during ongoing elections. However, the contract implementation lacks the necessary checks to ensure that cohort replacement aligns with ongoing governance activities.

The `replaceCohort` function is responsible for replacing a cohort with a new set of members. In this function, the old cohort is immediately deleted, and the new cohort is added. This process, while ensuring that only the new members are present in the cohort after replacement, does not account for potential ongoing elections or concurrent transactions.

```
function replaceCohort(address[] memory _newCohort, Cohort _coho
    external
    onlyRole(COHORT_REPLACER_ROLE)
{
    if (_newCohort.length != cohortSize) {
        revert InvalidNewCohortLength({cohort: _newCohort, cohor
    }

    // Delete the old cohort
    _cohort == Cohort.FIRST ? delete firstCohort : delete second

    // Add members of the new cohort
    for (uint256 i = 0; i < _newCohort.length; i++) {
        _addMemberToCohortArray(_newCohort[i], _cohort);
    }

    _scheduleUpdate();
    emit CohortReplaced(_newCohort, _cohort);
}
```

Recommended Mitigation Steps

One possible mitigation is to introduce a mechanism that prevents cohort replacement while an election is ongoing.

Assessed type

Invalid Validation

DZGoldman (Arbitrum) acknowledged

Low Risk and Non-Critical Issues

For this audit, 14 reports were submitted by wardens detailing low risk and non-critical issues. The **report highlighted below** by **MiloTruck** received the top score from the judge.

The following wardens also submitted reports: **ktg**, **berlin-101**, **HE1M**, **eierina**, **Udsen**, **0xprinc**, **hals**, **0xbepresent**, **arialblack14**, **Mirror**, **0xnev**, **nobody2018**, *and* **Sathish9098**.

Findings Summary

ID	Description	Severity
[L-01]	Governor contracts should prevent users from directly transferring ETH or tokens	Low
[L-02]	`areAddressArraysEqual()` isn't foolproof when both arrays have duplicate elements	Low
[L-03]	Missing duplicate checks in `L2SecurityCouncilMgmtFactory`'s `deploy()`	Low
[L-04]	Nominees excluded using `excludeNominee()` cannot be added back using `includeNominee()`	Low
[N-01]	Check that `_addressToRemove` and `_addressToAdd` are not equal in `_swapMembers()`	Non-Critical
[N-02]	Document how ties are handled for member elections	Non-Critical
[N-03]	`relay()` is not declared as `payable`	Non-Critical

ID	Description	Severity
[N-04]	Consider checking that `msg.value` is 0 in `_execute()` of governor contracts	Non-Critical
[N-05]	`topNominees()` could consume too much gas	Non-Critical
[N-06]	`_execute()` in `SecurityCouncilNomineeElectionGovernor` is vulnerable to blockchain re-orgs	Non-Critical

[L-01] Governor contracts should prevent users from directly transferring ETH or tokens

Openzeppelin's `GovernorUpgradeable` contract contains `receive()`, `onERC721Received()`, `onERC1155Received()` and `onERC1155BatchReceived()` to allow inheriting contracts to receive ETH and tokens.

However, this allows users to accidentally transfer their ETH/tokens to the governor contracts, which will then remain stuck until they are rescued by governance.

Recommendation

In `SecurityCouncilNomineeElectionGovernor`, `SecurityCouncilMemberRemovalGovernor` and `SecurityCouncilMemberElectionGovernor`, consider overriding these functions and making them revert. This prevents users from accidentally transferring ETH/tokens to the contracts.

[L-02] `areAddressArraysEqual()` isn't foolproof when both arrays have duplicate elements

The `areAddressArraysEqual()` function is used to check if `array1` and `array2` contain the same elements. It does so by checking that each element in `array1` exists in `array2`, and vice versa:

SecurityCouncilMgmtUpgradeLib.sol#L61-L85

```
        for (uint256 i = 0; i < array1.length; i++) {
            bool found = false;
```

```
            for (uint256 j = 0; j < array2.length; j++) {
                if (array1[i] == array2[j]) {
                    found = true;
                    break;
                }
            }
            if (!found) {
                return false;
            }
        }

        for (uint256 i = 0; i < array2.length; i++) {
            bool found = false;
            for (uint256 j = 0; j < array1.length; j++) {
                if (array2[i] == array1[j]) {
                    found = true;
                    break;
                }
            }
            if (!found) {
                return false;
            }
        }
```

However, this method isn't foolproof when both `array1` and `array2` contain duplicate elements. For example:

- `array1 = [1, 1, 2]`
- `array2 = [1, 2, 2]`

Even though both arrays are not equal, `areAddressArraysEqual()` will return true as they have the same length and all elements in one array exist in the other.

Recommendation

Consider checking that both arrays do not contain duplicate elements.

[L-03] Missing duplicate checks in `L2SecurityCouncilMgmtFactory`'s `deploy()`

In `L2SecurityCouncilMgmtFactory.sol`, the `deploy()` function only checks that every address in every cohort is an owner in `govChainEmergencySCSafe`:

[L2SecurityCouncilMgmtFactory.sol#L111-L121]

```
for (uint256 i = 0; i < dp.firstCohort.length; i++) {
    if (!govChainEmergencySCSafe.isOwner(dp.firstCohort[
        revert AddressNotInCouncil(owners, dp.firstCohor
    }
}

for (uint256 i = 0; i < dp.secondCohort.length; i++) {
    if (!govChainEmergencySCSafe.isOwner(dp.secondCohort
        revert AddressNotInCouncil(owners, dp.secondCoho
    }
}
```

However, there is no check to ensure that `firstCohort` and `secondCohort` do not contain any duplicates, or that any address in one cohort is not in the other. This makes it possible for the `SecurityCouncilManager` contract to be deployed with incorrect cohorts.

Recommendation

Consider checking the following:

- `firstCohort` and `secondCohort` do not contain any duplicates
- An address that is in `firstCohort` must not be in `secondCohort`, and vice versa.

[L-04] Nominees excluded using `excludeNominee()` cannot be added back using `includeNominee()`

In `SecurityCouncilNomineeElectionGovernor`, once nominees are excluded from the election by the nominee vetter using `excludeNominee()`, they cannot be added back using the `includeNominee()`.

This is because excluded nominees are not removed from the array of nominees, but are simply marked as excluded in the `isExcluded` mapping:

SecurityCouncilNomineeElectionGovernor.sol#L279-L280

```
            election.isExcluded[nominee] = true;
            election.excludedNomineeCount++;
```

Therefore, following check in `includeNominee()` will still fail when it is called for excluded nominees:

SecurityCouncilNomineeElectionGovernor.sol#L296-L298

```
            if (isNominee(proposalId, account)) {
                revert NomineeAlreadyAdded(account);
            }
```

This could become a problem if the nominee vetter accidentally calls `excludeNominee()` on the wrong nominee, or if there is some other legitimate reason a previously excluded nominee needs to be added back to the election.

[N-01] Check that `_addressToRemove` and `_addressToAdd` are not equal in `_swapMembers()`

In `_swapMembers()`, consider checking that `_addressToRemove` and `_addressToAdd` are not the same address:

SecurityCouncilManager.sol#L218-L229

```
        function _swapMembers(address _addressToRemove, address _add
            internal
            returns (Cohort)
        {
            if (_addressToRemove == address(0) || _addressToAdd == a
                revert ZeroAddress();
            }
    +       if (_addressToRemove == _addressToAdd) {
```

```
+            revert CannotSwapSameMembers();
+        }
         Cohort cohort = _removeMemberFromCohortArray(_addressToR
         _addMemberToCohortArray(_addressToAdd, cohort);
         _scheduleUpdate();
         return cohort;
     }
```

This would prevent scheduling unnecessary updates as there no changes to the security council members.

[N-02] Document how ties are handled for member elections

In the Arbitrum Constitution, there is no specification on how members are chosen in the event nominees are tied for votes.

Currently, `selectTopNominees()` simply picks the first 6 nominees after `LibSort.insertionSort()` is called, which means the nominee selected is random in the event they tie:

SecurityCouncilMemberElectionGovernorCountingUpgradeable.sol#L203-L217

```
        uint256[] memory topNomineesPacked = new uint256[](k);

        for (uint16 i = 0; i < nominees.length; i++) {
            uint256 packed = (uint256(weights[i]) << 16) | i;

            if (topNomineesPacked[0] < packed) {
                topNomineesPacked[0] = packed;
                LibSort.insertionSort(topNomineesPacked);
            }
        }

        address[] memory topNomineesAddresses = new address[](k)
        for (uint16 i = 0; i < k; i++) {
            topNomineesAddresses[i] = nominees[uint16(topNominee
        }
```

This could be confusing for users who expect tiebreaks to be handled in a deterministic manner (eg. whoever got the number of votes first).

Recommendation

Consider documenting how voting ties are handled in the Arbitrum Constitution to prevent confusion.

[N-03] `relay()` is not declared as `payable`

In `SecurityCouncilNomineeElectionGovernor`, although `relay()` makes calls with `AddressUpgradeable.functionCallWithValue()`, it is not declared as `payable`:

SecurityCouncilNomineeElectionGovernor.sol#L254-L261

```
    function relay(address target, uint256 value, bytes calldata
        external
        virtual
        override
        onlyOwner
    {
        AddressUpgradeable.functionCallWithValue(target, data, va
    }
```

This limits the functionality of `relay()`, as governance will not be able to send ETH to this contract and transfer the ETH to `target` in a single call to `relay()`.

Recommendation

Consider declaring `relay()` as `payable`:

SecurityCouncilNomineeElectionGovernor.sol#L254-L261

```
    function relay(address target, uint256 value, bytes calldata
        external
+       payable
        virtual
        override
        onlyOwner
    {
        AddressUpgradeable.functionCallWithValue(target, data, va
```

```
    }
```

This applies to the `relay()` function in `SecurityCouncilMemberElectionGovernor` and `SecurityCouncilMemberRemovalGovernor` as well.

[N-04] Consider checking that `msg.value` is 0 in `_execute()` of governor contracts

In Openzeppelin's `GovernorUpgradeable`, `execute()` is declared as `payable`:

GovernorUpgradeable.sol#L295-L300

```
    function execute(
        address[] memory targets,
        uint256[] memory values,
        bytes[] memory calldatas,
        bytes32 descriptionHash
    ) public payable virtual override returns (uint256) {
```

This makes it possible for users to accidentally transfer ETH to the governor contracts when calling `execute()`.

Recommendation

In `SecurityCouncilNomineeElectionGovernor`, `SecurityCouncilMemberRemovalGovernor` and `SecurityCouncilMemberElectionGovernor`, consider overriding `_execute()` and reverting if `msg.value` is not 0. This ensures that users cannot accidentally lose their ETH while calling `execute()`.

[N-05] `topNominees()` could consume too much gas

In `SecurityCouncilMemberElectionGovernorCountingUpgradeable.sol`, the `topNominees()` function is extremely gas-intensive due to the following reasons:

- `_compliantNominees()` copies the entire nominees array the storage of the `SecurityCouncilNomineeElectionGovernor` contract:

SecurityCouncilMemberElectionGovernorCountingUpgradeable.sol#L178

```
address[] memory nominees = _compliantNominees(proposalI
```

- `selectTopNominees()` iterates over all nominees and in the worst-case scenario, calls `LibSort.insertionSort()` in each iteration:

SecurityCouncilMemberElectionGovernorCountingUpgradeable.sol#L205-L212

```
for (uint16 i = 0; i < nominees.length; i++) {
    uint256 packed = (uint256(weights[i]) << 16) | i;

    if (topNomineesPacked[0] < packed) {
        topNomineesPacked[0] = packed;
        LibSort.insertionSort(topNomineesPacked);
    }
}
```

If the number of nominees is too large for an election, there is a significant chance that the `topNominees()` function will consume too much gas and revert due to an out-of-gas error.

If this occurs, member elections will be stuck permanently as proposals cannot be executed. This is because `_execute()` calls `topNominees()` to select the top nominees to replace the cohort in `SecurityCouncilManager`.

The number of nominees for an election is implicitly limited by the percentage of votes a contender needs to become a nominee. Currently, this is set to 0.2% which makes the maximum number of nominees 500. However, this also means that number of nominees could increase significantly should the percentage be decreased in the future.

[N-06] `_execute()` in

`SecurityCouncilNomineeElectionGovernor` is vulnerable to blockchain re-orgs

In `SecurityCouncilNomineeElectionGovernor`, when a user calls `execute()`, he only needs to provide the `proposalId` and current election index (in `callDatas`):

SecurityCouncilNomineeElectionGovernor.sol#L324-L329

```
function _execute(
    uint256 proposalId,
    address[] memory, /* targets */
    uint256[] memory, /* values */
    bytes[] memory callDatas,
    bytes32 /*descriptionHash*/
) internal virtual override {
```

This will then call `proposeFromNomineeElectionGovernor()` to start the member elections with the current nominees.

However, as the `_execute()`'s parameters do not contain any information about the list of qualified nominees, `execute()` is vulnerable to blockchain re-orgs. For example:

- Assume the following transactions are called in different blocks:
    - Block 1: Governance calls `excludeNominee()` to exclude nominee A from member elections.
    - Block 2: A user cals `execute()` to start member elections.
- A blockchain re-org occurs, block 1 is dropped and block 2 is kept.
- Now, when `execute()` is called in block 2, nominee A will still be in the list of compliant nominees, making him qualfiied for member elections.

Note that the risk of a blockchain re-org occurring is extremely small as Arbitrum cannot re-org unless the L1 itself re-orgs. Nevertheless, there is still a non-zero possibility of such a scenario occurring.

yahgwai (Arbitrum) commented:

> **L-01:** Acknowledge, won't fix.
> **L-02:** Confirm, fixed.
> **L-03:** Confirm, fixed in ts deploy script.
> **L-04:** Dispute, constitution doesn't say nominee vetter can change their mind.
> **N-01:** Acknowledge, could be a nice check, won't fix though.
> **N-02:** Confirm, added documentation.
> **N-03:** Acknowledge, don't forsee needing payable here, but it's possible to send in funds via receive as a workaround if absolutely necessary.
> **N-04:** Acknowledge, won't fix.
> **N-05:** Dispute, comments exist pointing this out already.
> **N-06:** Dispute, this isn't a vuln, it's just consistent with not calling includeNominee on time.

Gas Optimizations

For this audit, 13 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **LeoS** received the top score from the judge.

The following wardens also submitted reports: 0xAnah, JCK, Sathish9098, rektthecode, petrichor, Udsen, naman1778, SAAJ, dharma09, caventa, K42, *and* ernestognw.

Summary

	Issue	Instances	Gas Saved	
[G-01]	Use `calldata` instead of `memory`	4	-341 522	
[G-02]	The creation of an intermediary array can be avoided	1	-323 094	
[G-03]	Combine multiple for loop	2	-285 015	
[G-04]	Optimize array comparison	1	-136 722	
[G-05]	Set the number of optimizer runs individually	-	-	

The gas saved column simply adds up the evolution in the snapshot, using the method described in the next section.

Benchmark

A benchmark is performed on each optimization, using the tests snapshot provided by foundry. This snapshot is based on tests and therefore does not take into account all functions, this loss is accepted. But it is also subject to intrinsic variance. This means that some tests are not relevant to the comparison because they vary too much and are thus not taken into account. These are listed below:

- testNoopUpdate
- testRemoveOne
- testAddOne
- testUpdateCohort
- testCantDropBelowThreshhold

[G-01] Use `calldata` instead of `memory`

Using `calldata` instead of `memory` for function parameters can save gas if the argument is only read in the function. Only instances compilable with a simple type swap are considered valid.

4 instances

- SecurityCouncilManager.sol#L93
- SecurityCouncilManager.sol#L370
- SecurityCouncilNomineeElectionGovernor.sol#L103
- L2SecurityCouncilMgmtFactory.sol#L81 (only `ContractImplementations memory impls`)

Applying this optimisation, those changes appear in the snapshot:

```
testCantUpdateCohortWithADup() (gas: 20 (0.016%))
testE2E() (gas: -51950 (-0.062%))
testSecurityCouncilManagerDeployment() (gas: -47927 (-0.160%))
testNomineeElectionGovDeployment() (gas: -47927 (-0.160%))
testRemovalGovDeployment() (gas: -47927 (-0.160%))
testMemberElectionGovDeployment() (gas: -47927 (-0.160%))
testOnlyOwnerCanDeploy() (gas: -49346 (-0.196%))
testInvalidInit() (gas: -25509 (-0.364%))
testInitialization() (gas: -1263 (-0.626%))
```

```
testUpdateSecondCohort() (gas: -2014 (-0.657%))
testUpdateFirstCohort() (gas: -2014 (-0.657%))
testReplaceMemberInSecondCohort() (gas: -2034 (-0.744%))
testRotateMember() (gas: -2034 (-0.753%))
testReplaceMemberInFirstCohort() (gas: -2034 (-0.754%))
testAddMemberAffordances() (gas: -1892 (-0.758%))
testRemoveMember() (gas: -1892 (-0.872%))
testAddMemberToSecondCohort() (gas: -3926 (-1.109%))
testAddMemberToFirstCohort() (gas: -3926 (-1.119%))
Overall gas change: -341522 (-0.024%)
```

[G-02] The creation of an intermediary array can be avoided

Sometimes, it's not necessary to create an intermediate array to store values.

1 instance

- SecurityCouncilMgmtUtils.sol#L15-L36

In this case, an array of maximum size is created because we don't yet know what size the final array will be. This is not useful, as it's more efficient to keep this maximum-size array, fill it and then reduce its size using assembly.

The function can be changed like that:

```
function filterAddressesWithExcludeList(
address[]  memory input,
mapping(address => bool)  storage excludeList
)  internal  view  returns  (address[]  memory)  {
- /* */
+       address[]  memory output =  new  address[](input.length)
+       uint256 outputLength =  0;
+       for  (uint256 i =  0; i < input.length; i++)  {
+               if  (!excludeList[input[i]])  {
+                       output[outputLength]  = input[i];
+                       outputLength++;
+               }
+       }
+       assembly {
+               mstore(output, outputLength) //Set the appropria
+       }
+       return output;
```

```
}
```

Applying this optimisation, those changes appear in the snapshot:

```
testE2E() (gas: -31712 (-0.038%))
testSecurityCouncilManagerDeployment() (gas: -30294 (-0.101%))
testNomineeElectionGovDeployment() (gas: -30294 (-0.101%))
testRemovalGovDeployment() (gas: -30294 (-0.101%))
testMemberElectionGovDeployment() (gas: -30294 (-0.101%))
testOnlyOwnerCanDeploy() (gas: -30294 (-0.120%))
testInvalidInit() (gas: -30293 (-0.433%))
testTopNomineesGas() (gas: -109619 (-2.434%))
Overall gas change: -323094 (-0.023%)
```

[G-03] Combine multiple for loop

Whenever possible, it's best to avoid running several for loops one after the other. Most of the time, they can be combined. This avoids certain initialization and counting operations. It's even more interesting when you can take advantage of values that have already been calculated (in this case, a sum of lengths).

2 instances

- [L2SecurityCouncilMgmtFactory.sol#L107-L121]

```
+                uint256 totalCohortLength = dp.firstCohort.lengt
-        if (owners.length != (dp.firstCohort.length + dp.secondC
+        if  (owners.length !=  (totalCohortLength))  {
             revert InvalidCohortsSize(owners.length, dp.firstCoh
         }

-         for (uint256 i = 0; i < dp.firstCohort.length; i++) {
-             if (!govChainEmergencySCSafe.isOwner(dp.firstCohort
-                 revert AddressNotInCouncil(owners, dp.firstCoho
-             }
-         }

-         for (uint256 i = 0; i < dp.secondCohort.length; i++) {
-             if (!govChainEmergencySCSafe.isOwner(dp.secondCohor
-                 revert AddressNotInCouncil(owners, dp.secondCoh
```

```
-            }
-        }
+                for (uint256 i = 0; i < totalCohortLength; i++)
+                        address cohortMember;
+                        if (i < dp.firstCohort.length) {
+                                cohortMember = dp.firstCohort[i]
+                        }
+                        else {
+                                cohortMember = dp.secondCohort[i
+                        }
+                        if (!govChainEmergencySCSafe.isOwner(coh
+                                revert AddressNotInCouncil(owner
+                        }
+                }
```

Applying this optimisation, those changes appear in the snapshot:

```
testE2E() (gas: -38080 (-0.046%))
testSecurityCouncilManagerDeployment() (gas: -38605 (-0.129%))
testNomineeElectionGovDeployment() (gas: -38605 (-0.129%))
testRemovalGovDeployment() (gas: -38605 (-0.129%))
testMemberElectionGovDeployment() (gas: -38605 (-0.129%))
testOnlyOwnerCanDeploy() (gas: -38667 (-0.154%))
Overall gas change: -231167 (-0.016%)
```

- SecurityCouncilMemberSyncAction.sol#L60-L72

```
-         for (uint256 i = 0; i < _updatedMembers.length; i++) {
-             address member = _updatedMembers[i];
-             if (!securityCouncil.isOwner(member)) {
-                 _addMember(securityCouncil, member, threshold);
-             }
-         }
-
-         for (uint256 i = 0; i < previousOwners.length; i++) {
-             address owner = previousOwners[i];
-             if (!SecurityCouncilMgmtUtils.isInArray(owner, _upd
-                 _removeMember(securityCouncil, owner, threshold
-             }
-         }

+         for (uint256 i = 0; i < _updatedMembers.length || i < p
```

```
+            if (i < _updatedMembers.length) {
+                address member = _updatedMembers[i];
+                if (!securityCouncil.isOwner(member)) {
+                    _addMember(securityCouncil, member, thresho
+                }
+            }
+
+            if (i < previousOwners.length) {
+                address owner = previousOwners[i];
+                if (!SecurityCouncilMgmtUtils.isInArray(owner,
+                    _removeMember(securityCouncil, owner, thres
+                }
+            }
+        }
```

Applying this optimisation, those changes appear in the snapshot:

```
testNonces() (gas: -944 (-0.011%))
testE2E() (gas: -52904 (-0.064%))
Overall gas change: -53848 (-0.004%)
```

[G-04] Optimize array comparison

The following function is not well optimized. First of all, there's no need to check the array twice. This alone reduces the cost by half.

1 instance

- SecurityCouncilMgmtUpgradeLib.sol#L52-L88

```
function areAddressArraysEqual(address[] memory array1, address[
        public
        pure
        returns (bool)
{
        if (array1.length != array2.length) {
                return false;
        }
         for (uint256 i = 0; i < array1.length; i++) {
                bool found = false;
                for (uint256 j = 0; j < array2.length; j++) {
```

```
                        if (array1[i] == array2[j]) {
                                found = true;
                                break;
                        }
                }
                if (!found) {
                        return false;
                }
        }
-       for (uint256 i = 0; i < array2.length; i++) {
-               bool found = false;
-               for (uint256 j = 0; j < array1.length; j++) {
-                       if (array2[i] == array1[j]) {
-                               found = true;
-                               break;
-                       }
-               }
-               if (!found) {
-                       return false;
-               }
-       }
        return true;
```

Applying this optimisation, those changes appear in the snapshot:

```
testE2E() (gas: -136722 (-0.164%))
Overall gas change: -136722 (-0.010%)
```

But at this point the function is not yet perfectly optimized. it has a complexity of O(n^2). This can be increased by hashing the arrays, but the elements must be ordered. For this, quicksort is the best algorythm (this is obviously not recalled) (address can be compared by converting them to uint160).

```
function areAddressArraysEqual(address[] memory array1, address[
-    /* */
+       quicksort(array1);
+       quicksort(array2);
+       bytes32 hash1 = keccak256(abi.encode(array1));
+       bytes32 hash2 = keccak256(abi.encode(array2));
+       return hash1 == hash2;
+ }
```

In this way, the function would have a theoretical complexity of O(n log n). But in reality this will be more expensive for small arrays than the first alternative. Another way would be to introduce a hash table to compare elements more efficiently without looping (only one array need to be hashed), but this is again uneconomical on small arrays. That's why only a benchmark for the simplest optimization is provided. On tests, the other two worsen costs. But it's useful to know that they exist and could be implemented if this function were to become more important.

[G-05] Set the number of optimizer runs individually

Some development environments allow the number of optimizer runs to be defined individually for each contract. This is difficult to achieve in foundry. But hardhat is the best example. The main limitation to the number of runs in the case of contracts that will be heavily executed is the size of the deployment, which is limited. With foundry, a single contract completely blocks this number. Here, the maximum number is 1915 (set to 1900 in foundry) for the `SecurityCouncilNomineeElectionGovernor.sol` contract. By only setting this one to this value and greatly increase for the other, a huge saving can be achieved.

We therefore suggest moving to a development/deployment environment where this option is available. No contract in scope expects the limit, even for 100,000 runs.

This optimization certainly has the biggest impact, but as the tests are written with foundry, it's impossible to provide a benchmark in a short space of time.

yahgwai (Arbitrum) commented:

> All good findings, much appreciated.

> **G-01:** Acknowledge
> **G-02:** Acknowledge
> **G-03:** Acknowledge
> **G-04:** Confirm
> **G-05:** Nice info, thank you

Audit Analysis

For this audit, 10 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 **catellatech** received the top score from the judge.

The following wardens also submitted reports: 0xSmartContract, Sathish9098, hals, K42, berlin-101, MSK, yixxas, 0xnev, *and* kodyvim.

Summary

Arbitrum is a set of solutions designed to accelerate and make applications in blockchain more cost-effective. This is achieved by processing multiple transactions at once without incurring high costs. These solutions are based on Ethereum technology, a well-known blockchain platform.

Within the Arbitrum environment, there's a group called the "**Arbitrum DAO**" that makes decisions affecting two chains named "**Arbitrum One**" and "**Nova**". These decisions are crucial to ensure the security and proper functioning of these chains: When a new L2 chain is authorized by the DAO, the following steps should be carried out for the new chain to become DAO-governed:

1. Deploy a new UpgradeExecutor contract and a new Security Council on the new L2 chain.
2. Initialize the new L2 UpgradeExectutor with the L1 Timelock's aliased addressed and the new Security Council as its executors.
3. Ownership transfer: for a chain deployed whose contract deployment mirrors that of Arbitrum One and Arbitrum Nova (i.e, Nitro core contracts and token bridge contracts), the following ownership transfer should take place:

    - The L1 Upgrade Executor should be granted the following affordances:

        - L1 core contract Proxy Admin owner
        - L1 token bridge Proxy Admin owner
        - Rollup Admin owner
        - L1 Gateway Router owner
        - L1 Arb Custom Gateway Owner

- The new L2 Upgrade Executor should be granted the following affordances:
    - L2 token bridge Proxy Admin Owner
    - Chain Owner
    - Standard Arb-ERC20 Beacon Proxy owner

In addition, there's the "**Security Council**", consisting of 12 individuals from various organizations. This council takes on the responsibility of making quick decisions in the event of a significant security issue arising in the Arbitrum chains. They have the authority to implement urgent actions and resolve such problems. This council is divided into two groups, each of which is elected alternately every six months.

The process of selecting these groups is referred to as "elections" and is governed by a system of smart contracts on the chain. Terms like "contenders," "nominees," and "members" are used to describe the different stages and roles within this process.

Scope

- The engagement involved auditing twenty different Solidity Smart Contracts:
    - **SecurityCouncilManager.sol**: Serves as the definitive source of the current status for all security councils. It initiates updates across different chains and collaborates with the UpgradeExecRouteBuilder to craft the necessary payload.
    - **SecurityCouncilNomineeElectionGovernor.sol**: Empowers delegates to vote and select nominees from a broader pool of contenders. This module introduces a vetting period, allowing a trusted entity to include or exclude contenders based on their credibility.
    - **SecurityCouncilNomineeElectionGovernorCountingUpgradeable.sol**: This module handles the vote counting mechanism within the nominee governor system.
    - **SecurityCouncilNomineeElectionGovernorTiming.sol**: Provides timing-related functionalities for the nominee election governor module.
    - **SecurityCouncilMemberElectionGovernor.sol**: Enables delegates to vote for 6 candidates from a shortlist of nominees.

- **SecurityCouncilMemberElectionGovernorCountingUpgradeable.sol**: Manages the vote counting process within the member election governor module.
- **SecurityCouncilMemberRemovalGovernor.sol**: Facilitates the voting process for the removal of a candidate from the existing security council.
- **ArbitrumGovernorVotesQuorumFractionUpgradeable.sol**: Core functionality focused on counting only "votable" tokens.
- **ElectionGovernor.sol**: Provides shared functionalities that are utilized by various election governors.
- **UpgradeExecRouteBuilder.sol**: Constructs routes that target the upgrade executors on each respective chain.
- **SecurityCouncilMemberSyncAction.sol**: An action contract responsible for updating the member list of the Gnosis Safe.
- **SecurityCouncilMgmtUtils.sol**: Houses shared array utilities to enhance code efficiency.
- **Common.sol**: Contains shared data structures that are widely used.
- **L2SecurityCouncilMgmtFactory.sol**: Deploys contracts related to the election process.
- **GovernanceChainSCMgmtActivationAction.sol**: Activates the election process on Arbitrum One.
- **L1SCMgmtActivationAction.sol**: Initiates the election process on the L1 Ethereum chain.
- **NonGovernanceChainSCMgmtActivationAction.sol**: Activates the election process on Arbitrum Nova.
- **SecurityCouncilMgmtUpgradeLib.sol**: Contains shared utilities to streamline the management process.
- **KeyValueStore.sol**: Functions as a repository for values linked to specific keys, serving as external storage to prevent actions from directly utilizing state.
- **ActionExecutionRecord.sol**: Stores a record of executed actions for future reference.

Approach of the code base

During the analysis, our approach was to identify key contracts and their interactions within the Security Council management system. To determine which contract was the main one, we examined the hierarchy and responsibilities of the contracts provided. The main contract is the “Security Council Manager (SCM),” as it appears to be the central core of the system that coordinates and manages various governance functions related to the Security Council.

Starting from that main contract, we constructed to show how other contracts interact with it and contribute to different aspects of the governance process, such as member elections, activation on different chains, and action record management. Each contract plays a specific role and contributes to the overall operation of the system.

Click on the diagram links:

- Installation
- Arbitrum docs
- Slither
- Tests
- Scope
- draw.io
- Areas of concern

Analysis of the main contract

The central and most critical contracts within the project.

SecurityCouncilManager:

Architecture Description overview

The system architecture comprises various interconnected smart contracts that collaborate to manage and administer the Security Council’s governance process. Here is an overview of the components and their roles within the architecture::

Codebase Quality

Overall, we consider the quality of Arbitrum codebase to be excellent. The code appears to be very mature and well-developed. Details are explained below:

Codebase Quality Categories	Comments
Unit Testing	Codebase is well-tested it was great to see the protocol using Forundry framework.
Code Comments	Comments in general were solid. However is always room for improvement. Some areas could benefit from greater clarity in comments or explanations. Providing more detailed comments and documentation for complex or critical sections of the code can greatly enhance the codebase's overall readability and maintainability. This would not only help the current development team but also make it easier for future contributors to understand and build upon the existing code.
Documentation	The documentation for Arbitrum is very good.
Organization	Codebase is very mature and well organized with clear distinctions between the 20 contracts.

Systemic & Centralization Risks

The analysis provided highlights several significant systemic and centralization risks present in the present Arbitrum audit.

1. **dependency risk:**

    - It is observed that old and unsafe versions of Openzeppelin are used in the project, this should be updated to the latest one:

```
package.json#L69-L70
69: "@openzeppelin/contracts": "4.7.3",
70: "@openzeppelin/contracts-upgradeable": "4.7.3",
```

https://security.snyk.io/package/npm/@openzeppelin%2Fcontracts/4.9.3

2. The following is a list of quirks and/or potentially unexpected behaviors in the Arbitrum Governance implementation. General familiarity with the architecture is assumed:

    - **Abstain Vote** Voting "abstain" on a core-governor or governor proposal does not count as either a "for" or "against" vote, but does count towards reaching quorum (5% or 3% of votable tokens, respectively).
    - **Timelock vs Governor Execution** An operated queued in the core-governor-timelock or the treasury-governor-timelock can be executed permissionlessly on either its associated governor (typical) or on the timelock itself (atypical). The execution will be the same in either case, but in the later case, the governor's `ProposalExecuted` event will not be emitted.
    - **L1-to-L2 Message Fees in scheduleBatch** When executing a batch of operations on the L1ArbitrumTimelock, if more than one operation creates a retryable ticket, the full `msg.value` value will be forwarded to each one. For the execution to be successful, the `msg.value` should be set to `m` and the L1ArbitrumTimelock should be prefunded with at least `m * n` ETH, where `m` is the max out of the costs of each retryable ticket, and `n` is the number of retryable tickets created.
    - **Two L1 Proxy Admins** There are two L1 proxy admins - one for the governance contracts, once for the governed core Nitro contracts. Note that both proxy admins have the same owner (the DAO), and thus this has no material effect on the DAO's affordances.
    - **Non-excluded L2 Timelock** In the both treasury timelock and the DAO treasury can be transfered via treasury gov DAO vote; however, only ARB in the DAO treasury is excluded from the quorum numerator calculation. Thus, the DAO's ARB should ideally all be stored in the DAO Treasury. (Currently, the sweepReceiver in the TokenDistributor is set to the timelock, not the DAO treasury.)
    - **L2ArbitrumGovernoer onlyGovernance behavior** Typically, for a timelocked OZ governror, the `onlyGovernance` modifier ensures a call is made from the

timelock; in L2ArbitrumGoverner, the _executor() method is overriden such that `onlyGovernance` enforces a call from the governor contract itself. This ensures calls guarded by `onlyGovernance` go through the full core proposal path, as calls from the governor could only be sent via `relay`. See the code comment on `relay` in L2ArbitrumGoveror for more.

- **L2 Proposal Cancelation** There are two redundant affordances that the security council can use to cancel proposals in the L2 timelock: relaying through core governor, or using its `CANCELLER_ROLE` affordance. Additionally, the later affordances is granted directly to the security council, not the `UpgradeExecutor` (this is inconsistent with how `UpgradeExecutors` are generally used elsewhere.)
- **L1 Proposal Cancelation** The Security Council — not the L1 `UpgradeExecutor` — has the affordance to cancel proposals in the L1 timelock, inconsistent with how `UpgradeExecutors` are generally used elsewhere.

Test Coverage: the test coverage provide by Arbitrum is the 94%, however we recomende 100% of the tets coverage.

Conclusion

In general, Arbitrum project exhibits an interesting and well-developed architecture we believe the team has done a good job regarding the code. Additionally, it is recommended to improve the documentation and comments in the code to enhance understanding and collaboration among developers and auditors. It is also highly recommended that the team continues to invest in security measures such as mitigation reviews, audits, and bug bounty programs to maintain the security and reliability of the project.

Time Spent

A total of 5 days were spent to cover this audit and fully understand the flow of the contracts.

Time spent:
24 hours

yahgwai (Arbitrum) acknowledged

Disclosures

C4 is an open organization governed by participants in the community.

C4 Audits incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Audit 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