MiloTruck

Arbitrum Security Council Elections

Security Review Report

August, 2023

Table of Contents

Table of Contents **1**

Introduction **3**

- About MiloTruck 3
- Disclaimer 3

Executive Summary **4**

- About Arbitrum Security Council Elections 4
- Repository Details 4
- Scope 4
- Issues Found 4

Findings **5**

- Summary 5
- High Severity Findings 6
  - H-01: Signatures can be replayed in `castVoteWithReasonAndParamsBySig()` to use up more votes than a user intended 6
- Medium Severity Findings 8
  - M-01: Missing `__Governor_init()` call in the `initialize()` function of `SecurityCouncilMemberRemovalGovernor` 8
  - M-02: `electionToTimestamp()` might return incorrect timestamps depending on the day of the first election 10
  - M-03: `setFullWeightDuration()` can be called while a member election is ongoing 13
- Low Severity Findings 15
  - L-01: `SecurityCouncilManager`'s `initialize()` function contains a gas bomb 15
  - L-02: Governance could accidentally DOS member elections by setting `_votingPeriod` less than `fullWeightDuration` 17
  - L-03: Consider checking that `msg.value` is 0 in `_execute()` of governor contracts 19
  - L-04: Governor contracts should prevent users from directly transferring ETH or tokens 19
  - L-05: Governance can DOS elections by setting `votingDelay` or `votingPeriod` more than `type(uint64).max` 20
  - L-06: `areAddressArraysEqual()` isn't foolproof when both arrays have duplicate elements 21
  - L-07: Missing duplicate checks in `L2SecurityCouncilMgmtFactory`'s `deploy()` 22
  - L-08: `topNominees()` could consume too much gas 23
  - L-09: Nominees excluded using `excludeNominee()` cannot be added back using `includeNominee()` 24
- Non-Critical Findings 25
  - N-01: Check that `_addressToRemove` and `_addressToAdd` are not equal in `_swapMembers()` 25
  - N-02: Document how ties are handled for member elections 26
  - N-03: `relay()` is not declared as `payable` 27

Introduction

About MiloTruck

MiloTruck is an independent security researcher who specializes in smart contract audits. Having won multiple audit contests, he is currently one of the top wardens on Code4rena.

For security consulting, reach out to him on Twitter - *@milotruck*

Disclaimer

A smart contract security review **can never prove the complete absence of vulnerabilities**. Security reviews are a time, resource and expertise bound effort to find as many vulnerabilities as possible. However, they cannot guarantee the absolute security of the protocol in any way.

Executive Summary

This review was completed as part of an audit contest on Code4rena.

About Arbitrum Security Council Elections

Arbitrum's Security Council is a group of individuals who are responsible for addressing risks to the Arbitrum ecosystem through the selective application of emergency actions and non-emergency actions.

They are a 12 member council from independent organizations that hold the ability to conduct emergency actions to the Arbitrum chains. The 12 member council is separated into 2 cohorts which are elected alternatively every 6 months.

This codebase includes the subsystem of Arbitrum Governance that allows these cohorts to be managed and elected through an on-chain suite of smart contracts written in Solidity.

Repository Details

Repository	https://github.com/code-423n4/2023-08-arbitrum
Commit Hash	7dcdab81e760fd328e63422d7d7a24a835cef1e7
Language	Solidity

Scope

The scope of this review can be found here.

Issues Found

Severity	Count
High	1
Medium	3
Low	9
Non-Critical	3

Findings

Summary

ID	Description	Severity
H-01	Signatures can be replayed in castVoteWithReasonAndParamsBySig() to use up more votes than a user intended	High
M-01	Missing __Governor_init() call in the initialize() function of SecurityCouncilMemberRemovalGovernor	Medium
M-02	electionToTimestamp() might return incorrect timestamps depending on the day of the first election	Medium
M-03	setFullWeightDuration() can be called while a member election is ongoing	Medium
L-01	SecurityCouncilManager's initialize() function contains a gas bomb	Low
L-02	Governance could accidentally DOS member elections by setting _votingPeriod less than fullWeightDuration	Low
L-03	Consider checking that msg.value is 0 in _execute() of governor contracts	Low
L-04	Governor contracts should prevent users from directly transferring ETH or tokens	Low
L-05	Governance can DOS elections by setting votingDelay or votingPeriod more than type(uint64).max	Low
L-06	areAddressArraysEqual() isn't foolproof when both arrays have duplicate elements	Low
L-07	Missing duplicate checks in L2SecurityCouncilMgmtFactory's deploy()	Low
L-08	topNominees() could consume too much gas	Low
L-09	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

High Severity Findings

H-01: Signatures can be replayed in `castVoteWithReasonAndParamsBySig()` to use up more votes than a user intended

Bug Description

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.

Link to PoC

Recommended Mitigation

Consider adding protection against signature replay 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.

Medium Severity Findings

M-01: Missing `__Governor_init()` call in the `initialize()` function of `SecurityCouncilMemberRemovalGovernor`

Bug Description

The `SecurityCouncilMemberRemovalGovernor` contract inherits Openzeppelin's `GovernorUpgradeable`:

SecurityCouncilMemberRemovalGovernor.sol#L17-L19

```
contract SecurityCouncilMemberRemovalGovernor is
    Initializable,
    GovernorUpgradeable,
```

However, in its `initialize()` function, `__Governor_init()` is never called. `__Governor_init()` is used to initialize the `GovernorUpgradeable` contract, which sets `_name`, `_HASHED_NAME` and `_HASHED_VERSION`:

GovernorUpgradeable.sol#L79-L82

```
    function __Governor_init(string memory name_) internal onlyInitializing {
        __EIP712_init_unchained(name_, version());
        __Governor_init_unchained(name_);
    }
```

GovernorUpgradeable.sol#L84-L86

```
    function __Governor_init_unchained(string memory name_) internal onlyInitializing {
        _name = name_;
    }
```

draft-EIP712Upgradeable.sol#L54-L59

```
    function __EIP712_init_unchained(string memory name, string memory version) internal
onlyInitializing {
        bytes32 hashedName = keccak256(bytes(name));
        bytes32 hashedVersion = keccak256(bytes(version));
        _HASHED_NAME = hashedName;
        _HASHED_VERSION = hashedVersion;
    }
```

As such, after `SecurityCouncilMemberRemovalGovernor` is deployed and initialized, `_HASHED_NAME` and `_HASHED_VERSION` will still be `bytes32(0)`. These values are used to build the domain separator when verifying signatures:

draft-EIP712Upgradeable.sol#L64-L66

```
    function _domainSeparatorV4() internal view returns (bytes32) {
        return _buildDomainSeparator(_TYPE_HASH, _EIP712NameHash(), _EIP712VersionHash());
    }
```

draft-EIP712Upgradeable.sol#L101-L103

```
    function _EIP712NameHash() internal virtual view returns (bytes32) {
        return _HASHED_NAME;
    }
```

draft-EIP712Upgradeable.sol#L111-L113

```
    function _EIP712VersionHash() internal virtual view returns (bytes32) {
        return _HASHED_VERSION;
    }
```

This becomes problematic when calling functions such as `castVoteBySig()` and `castVoteWithReasonAndParamsBySig()`, which relies on signatures for voting.

Impact

Since `__Governor_init()` is never called, `castVoteBySig()` and `castVoteWithReasonAndParamsBySig()` will revert for signatures generated using the `name()` and `version()` functions. This could break the functionality of frontends/contracts that rely on these functions to integrate with the protocol.

Note that this also violates the EIP-712 standard as the name and version parameters are incorrectly set to `bytes32(0)` when verifying signatures.

Recommended Mitigation

Call `__Governor_init()` in the contract's `initialize()` function:

SecurityCouncilMemberRemovalGovernor#L69-L76

```
    ) public initializer {
+       __Governor_init("SecurityCouncilMemberRemovalGovernor");
        __GovernorSettings_init(_votingDelay, _votingPeriod, _proposalThreshold);
        __GovernorCountingSimple_init();
        __GovernorVotes_init(_token);
        __ArbitrumGovernorVotesQuorumFraction_init(_quorumNumerator);
        __GovernorPreventLateQuorum_init(_minPeriodAfterQuorum);
        __ArbitrumGovernorProposalExpirationUpgradeable_init(_proposalExpirationBlocks);
        _transferOwnership(_owner);
```

M-02: `electionToTimestamp()` might return incorrect timestamps depending on the day of the first election

Bug Description

For nominee elections, election dates are determined using the the `electionToTimestamp()` function in the `SecurityCouncilNomineeElectionGovernorTiming` module.

When `SecurityCouncilNomineeElectionGovernor` is initialized after deployment, the call to `__SecurityCouncilNomineeElectionGovernorTiming_init()` stores the first election date. Afterwards, `electionToTimestamp()` will provide the next election timestamp based on the number of elections passed:

SecurityCouncilNomineeElectionGovernorTiming.sol#L75-L94

```
    function electionToTimestamp(uint256 electionIndex) public view returns (uint256) {
        // subtract one to make month 0 indexed
        uint256 month = firstNominationStartDate.month - 1;

        month += 6 * electionIndex;
        uint256 year = firstNominationStartDate.year + month / 12;
        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
        });
    }
```

As seen from above, `electionToTimestamp()` works by adding 6 months for every passed election, and then converting the date to a timestamp through Solady's `dateTimeToTimestamp()`.

However, this approach does not account for months that have less days than others.

For example, if the first election was scheduled on 31 August, the next election would be 31 February according to the formula above. However, as February doesn't have 31 days, the `day` parameter is outside the range supported by `dateTimeToTimestamp()`, resulting in undefined behavior:

DateTimeLib.sol#L131-L133

```
    /// Note: Inputs outside the supported ranges result in undefined behavior.
    /// Use {isSupportedDateTime} to check if the inputs are supported.
    function dateTimeToTimestamp(
```

Therefore, `dateTimeToTimestamp()` will return an incorrect timestamp.

Impact

If the first election starts on the 29th to 31st day of the month, `dateTimeToTimestamp()` could potentially return an incorrect timestamp for subsequent elections.

Proof of Concept

Assume that the first election is brought forward from 15 September 2023 to 31 August 2023. Every alternate election will now be held in February, which creates two problems:

1. The election date for one cohort will not be fixed

If the current year is a leap year, the election that was supposed to be held in February will be one day earlier than a non-leap year. For example:

- Since 2024 is a leap year, the second election will be on 2 March 2024.
- Since 2025 is not a leap year, the fourth election will be on 3 March 2025.

This becomes a problem as the Arbitrum Constitution states a specific date for the two elections in a year, which is not possible if the scenario above occurs.

2. One term is a few days shorter for a cohort

As mentioned above, if the start date was 31 August 2023, the fourth election will be on 3 March 2025. However, if the first election was on 3 September 2023, the fourth election would still be on 3 March 2025.

This means that the election starts three days later for the scenario above, making the term for one cohort a few days longer than intended.

Link to PoC

Recommended Mitigation

Ensure that `_firstNominationStartDate.day` is never above 28:

SecurityCouncilNomineeElectionGovernorTiming.sol#L41-L48

```diff
-       if (!isSupportedDateTime) {
+       if (!isSupportedDateTime || _firstNominationStartDate.day > 28) {
            revert InvalidStartDate(
                _firstNominationStartDate.year,
                _firstNominationStartDate.month,
                _firstNominationStartDate.day,
                _firstNominationStartDate.hour
            );
        }
```

Additionally, consider storing `startTimestamp` instead of `firstNominationStartDate`. With the first election's timestamp, subsequent election timestamps can be calculated using `DateTimeLib.addMonths()` instead:

```
    function electionToTimestamp(uint256 electionIndex) public view returns (uint256) {
        return DateTimeLib.addMonths(startTimestamp, electionIndex * 6);
    }
```

Using `addMonths()` ensures that election dates are always fixed, even in the scenario mentioned above.

M-03: `setFullWeightDuration()` can be called while a member election is ongoing

Bug Description

In `SecurityCouncilMemberElectionGovernorCountingUpgradeable`, `fullWeightDuration` (which is the duration where a user's votes has full weightage) can be set using `setFullWeightDuration()`:

SecurityCouncilMemberElectionGovernorCountingUpgradeable.sol#L77-L84

```
    function setFullWeightDuration(uint256 newFullWeightDuration) public onlyGovernance {
        if (newFullWeightDuration > votingPeriod()) {
            revert FullWeightDurationGreaterThanVotingPeriod(newFullWeightDuration, votingPeriod());
        }

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

`fullWeightDuration` is then used to calculate the weightage of a user's votes in `votesToWeight()`:

SecurityCouncilMemberElectionGovernorCountingUpgradeable.sol#L247-L255

```
        // Between the fullWeightVotingDeadline and the proposalDeadline each vote will have weight
linearly decreased by time since fullWeightVotingDeadline
        // slope denominator
        uint256 decreasingWeightDuration = endBlock - fullWeightVotingDeadline_;
        // slope numerator is -votes, slope denominator is decreasingWeightDuration, delta x is
blockNumber - fullWeightVotingDeadline_
        // y intercept is votes
        uint256 decreaseAmount =
            votes * (blockNumber - fullWeightVotingDeadline_) / decreasingWeightDuration;
        // subtract the decreased amount to get the remaining weight
        return _downCast(votes - decreaseAmount);
```

Where:

- `fullWeightVotingDeadline_` is equal to `startBlock + fullWeightDuration`.

However, as there is no restriction on when `setFullWeightDuration()` can be called, it could potentially be unfair for voters if `fullWeightDuration` is increased or decreased while an election is ongoing. For example:

- Assume the following:
  - `startBlock = 1000000`, `endBlock = 2000000`, `fullWeightDuration = 500000`
  - `block.timestamp` is currently `1625000`
- Alice calls `castVote()` to vote for a nominee:
  - As a quarter of `decreasingWeightDuration` has passed, her votes have a 75% weightage.
- Governance suddenly calls `setFullWeightDuration()` to set `fullWeightDuration` to `750000`.
- `block.timestamp` is now within the full weight duration, meaning that users who vote now will have a 100% weightage. However, Alice has already voted, and cannot undo her votes.

In the scenario above, future voters will have a larger weight than Alice for the same amount of votes, making it unfair for her.

Impact

If `setFullWeightDuration()` is called during an election, users who have already voted will be unfairly affected as their votes will have more/less weight than they should.

Given that `setFullWeightDuration()` is called by governance, which has to be scheduled through timelocks, it might be possible for governance to accidentally schedule a call that updates `fullWeightDuration` while an election is ongoing.

Recommended Mitigation

Consider allowing `setFullWeightDuration()` to be called only when there isn't an election ongoing.

One way of knowing when an election is ongoing is to track when the `proposeFromNomineeElectionGovernor()` and `_execute()` functions are called, which mark the start and end of an election respectively.

Low Severity Findings

L-01: `SecurityCouncilManager`'s `initialize()` function contains a gas bomb

Bug Description

In `SecurityCouncilManager.sol`, the `initialize()` function calls `_addSecurityCouncil()` in a loop to add security councils individually:

SecurityCouncilManager.sol#L118-L120

```
        for (uint256 i = 0; i < _securityCouncils.length; i++) {
            _addSecurityCouncil(_securityCouncils[i]);
        }
```

`_addSecurityCouncil()` performs some checks, which includes ensuring the new security council (`_securityCouncilData`) isn't already added, before adding it to the `securityCouncils` array:

SecurityCouncilManager.sol#L251-L262

```
        for (uint256 i = 0; i < securityCouncils.length; i++) {
            SecurityCouncilData storage existantSecurityCouncil = securityCouncils[i];

            if (
                existantSecurityCouncil.chainId == _securityCouncilData.chainId
                    && existantSecurityCouncil.securityCouncil ==
_securityCouncilData.securityCouncil
            ) {
                revert SecurityCouncilAlreadyInRouter(_securityCouncilData);
            }
        }

        securityCouncils.push(_securityCouncilData);
```

However, as the duplicate check loops over all elements in the `securityCouncils` storage array, `_addSecurityCouncil()` will consume a lot of gas whenever it is called.

As it is called repeatedly in `initialize()`, there is a significant chance that `initialize()` might consume too much gas when called, making it revert due to an out-of-gas error.

The contract declares a maximum limit of 500 security councils to mitigate this:

SecurityCouncilManager.sol#L67

```
    uint256 public immutable MAX_SECURITY_COUNCILS = 500;
```

However, this is insufficient as calling `initialize()` with 500 security councils will still read from storage 125,250 times, which will still consume a huge amount of gas.

Impact

If `SecurityCouncilManager` is initialized with a large number of security councils, the `initialize()` function might not be executable due to consuming too much gas.

Recommended Mitigation

Consider performing all checks in `initialize()` and pushing to the `securityCouncils` array directly, instead of calling `_addSecurityCouncil()`. This might reduce gas consumption significantly as the iteration is performed over the array stored in memory, thereby avoiding reading from storage.

L-02: Governance could accidentally DOS member elections by setting `_votingPeriod` less than `fullWeightDuration`

Bug Description

In `SecurityCouncilMemberElectionGovernorCountingUpgradeable`, `setFullWeightDuration()` has a check to ensure that `fullWeightDuration` is more than the voting period:

SecurityCouncilMemberElectionGovernorCountingUpgradeable.sol#L77-L84

```
    function setFullWeightDuration(uint256 newFullWeightDuration) public onlyGovernance {
        if (newFullWeightDuration > votingPeriod()) {
            revert FullWeightDurationGreaterThanVotingPeriod(newFullWeightDuration, votingPeriod());
        }

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

However, the `setVotingPeriod()` function in Openzeppelin's `GovernorSettingsUpgradeable` module doesn't ensure that `_votingPeriod` is above `fullWeightDuration`. This means that governance could accidentally cause `fullWeightDuration` to be greater than `_votingPeriod` by calling `setVotingPeriod()` to decrease the voting period.

Should this occur, `votesToWeight()` will revert due to an arithmetic underflow when performing the following calculation:

SecurityCouncilMemberElectionGovernorCountingUpgradeable.sol#L241-L249

```
        // Between proposalSnapshot and fullWeightVotingDeadline all votes will have
100% weight - each vote has weight 1
        uint256 fullWeightVotingDeadline_ = fullWeightVotingDeadline(proposalId);
        if (blockNumber <= fullWeightVotingDeadline_) {
            return _downCast(votes);
        }

        // Between the fullWeightVotingDeadline and the proposalDeadline each vote will
have weight linearly decreased by time since fullWeightVotingDeadline
        // slope denominator
        uint256 decreasingWeightDuration = endBlock - fullWeightVotingDeadline_;
```

Where:

- `endBlock` is equal to `startBlock + _votingPeriod`.
- `fullWeightVotingDeadline_` is equal to `startBlock + fullWeightDuration`.

Since `votesToWeight()` is used to determine the weightage of votes in `_countVote()`, all voting functions (eg. `castVote()`) will always revert once `fullWeightVotingDeadline_` has passed, causing all voting to be DOSed.

Impact

Governance could accidentally DOS member elections by reducing the voting period below `fullWeightDuration` using `setVotingPeriod()`.

This could occur if `fullWeightDuration` is initially equal to `votingPeriod` (votes have 100% weightage during the entire voting period), and governance decides to reduce the voting period to a shorter duration.

Given that `setFullWeightDuration()` is also called by governance, and has to be scheduled through timelocks, it might not be possible for governance to call `setFullWeightDuration()` to reduce `fullWeightDuration` in time after realizing the DOS has occurred.

Recommended Mitigation

In the `SecurityCouncilMemberElectionGovernor` contract, consider overriding the `setVotingPeriod()` function to ensure that the new voting period is always greater than `fullWeightDuration`. For example:

```
function setVotingPeriod(uint256 newVotingPeriod) public override onlyGovernance {
    if (newVotingPeriod <= fullWeightDuration) {
        revert NewVotingPeriodLessThanFullWeightDuration();
    }

    _setVotingPeriod(newVotingPeriod);
}
```

L-03: Consider checking that `msg.value` is 0 in `_execute()` of governor contracts

Bug Description

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()`.

Recommended Mitigation

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()`.

L-04: Governor contracts should prevent users from directly transferring ETH or tokens

Bug Description

Openzeppelin's `GovernorUpgradeable` contract contains the `receive()`, `onERC721Received()`, `onERC1155Received()` and `onERC1155BatchReceived()` functions 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.

Recommended Mitigation

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

L-05: Governance can DOS elections by setting `votingDelay` or `votingPeriod` more than `type(uint64).max`

Bug Description

In the `propose()` function of Openzeppelin's `GovernorUpgradeable` contract, `votingDelay` and `votingPeriod` are cast from `uint256` to `uint64` safely:

[GovernorUpgradeable.sol#L271-L272]

```
        uint64 snapshot = block.number.toUint64() + votingDelay().toUint64();
        uint64 deadline = snapshot + votingPeriod().toUint64();
```

Therefore, if either of these values are set to above `type(uint64).max`, `propose()` will revert.

Recommended Mitigation

In the `SecurityCouncilNomineeElectionGovernor`, `SecurityCouncilMemberRemovalGovernor` and `SecurityCouncilMemberElectionGovernor` contracts, consider overriding the `setVotingDelay()` and `setVotingPeriod()` functions to check that `votingDelay` and `votingPeriod` are not set to values above `type(uint64).max`.

L-06: `areAddressArraysEqual()` isn't foolproof when both arrays have duplicate elements

Bug Description

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.

Recommended Mitigation

Consider checking that both arrays do not contain duplicate elements.

L-07: Missing duplicate checks in `L2SecurityCouncilMgmtFactory`'s `deploy()`

Bug Description

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[i])) {
                revert AddressNotInCouncil(owners, dp.firstCohort[i]);
            }
        }

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

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.

Recommended Mitigation

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-08: `topNominees()` could consume too much gas

Bug Description

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(proposalId);
```

- `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.

Impact

Member elections might 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 the number of nominees could increase significantly should the percentage be decreased in the future.

L-09: Nominees excluded using `excludeNominee()` cannot be added back using `includeNominee()`

Bug Description

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);
        }
```

Impact

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.

Non-Critical Findings

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 _addressToAdd)
        internal
        returns (Cohort)
    {
        if (_addressToRemove == address(0) || _addressToAdd == address(0)) {
            revert ZeroAddress();
        }
+       if (_addressToRemove == _addressToAdd) {
+           revert CannotSwapSameMembers();
+       }
        Cohort cohort = _removeMemberFromCohortArray(_addressToRemove);
        _addMemberToCohortArray(_addressToAdd, cohort);
        _scheduleUpdate();
        return cohort;
    }
```

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

N-02: Document how ties are handled for member elections

Bug Description

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(topNomineesPacked[i])];
        }
```

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

Recommended Mitigation

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

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

Bug Description

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

SecurityCouncilNomineeElectionGovernor.sol#L254-L261

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

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()`.

Recommended Mitigation

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

SecurityCouncilNomineeElectionGovernor.sol#L254-L261

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

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