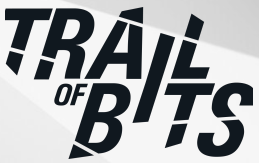

Offchain Labs, Security Council Elections

Security Assessment (Summary Report)

August 9, 2023

Prepared for:
Harry Kalodner, Steven Goldfeder, and Ed Felten
Offchain Labs

Prepared by: **Simone Monica and Jaime Iglesias**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Executive Summary

Engagement Overview

Offchain Labs engaged Trail of Bits to review the security of Security Council Elections contracts. The Security Council is a committee of 12 members who are signers of a multi-sig wallet, which has powers to perform certain Emergency Actions and Non-Emergency Actions. The Security Council Elections contracts allow the start of an election every 6 months to replace one cohort of the Security Council which corresponds to 6 of the 12 members.

A team of two consultants conducted the review from July 24 to July 28, for a total of two engineer-weeks of effort. Our testing efforts focused on the review of the contracts' behavior, which we found to be in line with the descriptions in the Arbitrum Constitution and the proposal forum post. With full access to source code and documentation, we performed static and dynamic testing, using automated and manual processes.

Observations and Impact

Most findings are of informational severity and do not pose any immediate risk; however, the changes made in PR #150 introduced a denial-of-service vector (TOB-ASC-3) that could prevent any future elections from being proposed unless a contract upgrade is made.

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
Medium	1
Low	1
Informational	3

CATEGORY BREAKDOWN

Category	*Count*
Denial of Service	1
Patching	1
Undefined Behavior	3

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Mary O'Brien, Project Manager
mary.obrien@trailofbits.com

The following engineers were associated with this project:

Jaime Iglesias, Consultant
jaime.iglesias@trailofbits.com

Simone Monica, Consultant
simone.monica@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
July 21, 2023	Pre-project kickoff call
July 31, 2023	Delivery of report draft
July 31, 2023	Report readout meeting
August 9, 2023	Delivery of summary report

Project Goals

The engagement was scoped to provide a security assessment of the Security Council Elections contract for Offchain Labs. Specifically, we sought to answer the following non-exhaustive list of questions:

- Do the contracts implement the behavior described in the Arbitrum Constitution and in the proposal forum post?
- Does the contract respect the election's timeline?
- Can anyone become a member of the Security Council?
- Is the linearly decreasing voting weight system for the member election implemented correctly?
- Does the action contract for updating security council members follow the action contract guidelines?
- Are there any denial-of-service risks?

Project Targets

The engagement involved a review and testing of the following target.

Security Council Elections

Repository	https://github.com/ArbitrumFoundation/governance
Version	a2b308f241991c71940672c71277429c79af47ee 27e66586af4e56d9ced3d7f06c3bbeb2bb2e89ce
Type	Solidity
Platform	Arbitrum

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- **Nominee election**: The `SecurityCouncilNomineeElectionGovernor` contract represents the first step in the election process. Anyone can submit themselves as contenders, and if they receive 0.2% of votes, they become a nominee and therefore become eligible to be voted for at the member election (the next step of the cohort election).

  We manually reviewed this contract for the correct timeline of the election process: for example, that an election can be created only every six months; that the nominee election step lasts for seven days (note that this time is a configurable parameter taken at initialization); and that the election is executed only after the amount of time allotted for the Foundation to carry out compliance checks has passed. We reviewed the access controls checks for correctness—in particular, that anyone can create and execute an election and that only the `nomineeVetter` can include or exclude nominees during the compliance check by the Foundation.

  Finally, we implemented some differential fuzzing between the `electionTimestamp` function and Solady's `DateTimeLib` to check the accuracy of the calculation of the timestamp of the next election (see appendix C for details).

- **Member election**: The `SecurityCouncilMemberElectionGovernor` contract implements the final step in the election process, in which the top six nominees become members of the Security Council.

  We manually reviewed the timeline of this final step for correctness and to assess whether the linearly decreasing voting weight system works as intended. Additionally, we differentially fuzzed the Solady's `insertionSort` function against an unoptimized version of the algorithm (see appendix C for details).

- **Member removal**: The `SecurityCouncilMemberRemovalGovernor` contract allows the DAO to vote for the removal of a member prior to the end of their term.

  We manually reviewed that the proposal can be initiated by anyone, that it can only call `removeMember` in the `SecurityCouncilManager` once executed, and that the address selected for removal is effectively a member of one of the two cohorts.

- **Security council manager**: The `SecurityCouncilManager` contract, as the name implies, contains all of the functionality related to the management of the security

council, including removing and adding members, replacing cohorts, and adding security councils of the different chains.

We manually reviewed the code to assess whether the functionality is correct (e.g., to assess whether a member can be in two cohorts at the same time). However, note that the manager has more freedom than the election system when it comes to managing the cohorts, and therefore some of its operations do not contain safety checks that are present in other parts of the election system.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- The Offchain Labs team provided us with documentation outlining the election process along with some notes on known risks and caveats of the system's design. Below are pertinent excerpts from this documentation:

  > *It's possible for Core Governance and the elections to schedule the same operation in timelocks, and therefore block each other. However the DAO would need to do this on purpose, and against their Constitution. If the DAO is willing to make proposals against their Constitution then they can arbitrarily change all the contracts, so a clashing operation is not considered an issue.*
  >
  > *Although anyone can execute the results of an election, the contracts do not enforce that an election is actually executed. This can mean that elections could in theory be executed out of order, resulting in an inconsistent state. If an election is not executed before the next one begins they are not expected to produce a consistent state.*
  >
  > - Note that PR #150 aims to address this issue.
  >
  > *The Security Council can create updates in the SecurityCouncilManager that could cause an election execution to fail. They are expected to take care when making these kinds of updates not to cause that issue. Since the Security Council can technically change all the contracts, a Council that is causing disruption can already do so in much wider ways.*
  >
  > *An additional method "rotateMember" is provided that does the same as "replaceMember". Although this method has the same functionality as "replaceMember" it has a different semantic meaning as it should be used to change the key of an existing member, rather than replace a member with a different one.*

- The deployment and activation scripts were only superficially reviewed, as these were lower priority for the client. Instead, we opted to focus our efforts on the higher priority targets.

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	The ISecurityCouncilUpgradeExectutor interface is unused	Undefined Behavior	**Informational**
2	SecurityCouncilMemberElectionGovernor contract doesn't disable initializers	Undefined Behavior	**Informational**
3	Failing cohort elections can block future elections	Denial of Service	**Medium**
4	Project dependencies contain vulnerabilities	Patching	**Informational**
5	Security council member can be removed unexpectedly	Undefined Behavior	**Low**

Detailed Findings

1. The ISecurityCouncilUpgradeExectutor interface is unused

Severity: **Informational**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-ASC-1
Target: `src/security-council-mgmt/interfaces/ISecurityCouncilUpgradeExectutor.sol`	

Description

The `ISecurityCouncilUpgradeExectutor` interface is unused, which causes confusion when reviewing the code. Additionally, the contract that should implement this interface may not adhere to the function declaration, which could become an issue if a third-party contract expects the contract to implement this interface.

```
interface ISecurityCouncilUpgradeExectutor {
     function perform(address[] memory _membersToAdd, address[] memory
_membersToRemove) external;
 }
```

Figure 1.1: The ISecurityCouncilUpgradeExectutor interface in ISecurityCouncilUpgradeExectutor.sol#L4-L6

Based on the interface's name, we believe that it should be implemented by the `SecurityCouncilMemberSyncAction` contract. However, note that the perform function takes different argument' types.

```
contract SecurityCouncilMemberSyncAction {
    ...
    function perform(address _securityCouncil, address[] memory _updatedMembers)
external {
```

Figure 1.2: The `perform` function header in SecurityCouncilMemberSyncAction.sol#L21

Recommendations

Short term, consider removing or implementing the interface in the appropriate contract.

2. SecurityCouncilMemberElectionGovernor contract doesn't disable initializers

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-ASC-2
Target: `src/security-council-mgmt/interfaces/SecurityCouncilMemberElectionGovernor.sol`	

Description
The `SecurityCouncilMemberElectionGovernor` contract, which is used behind a proxy, does not disable initializers in the constructor. This causes inconsistency with the other contracts that disable initializers and could result in unexpected behavior.

Note that in this particular case, the contract does not contain "critical" functionality that would allow a malicious actor to steal funds or destroy the system; however, future contract upgrades may have this functionality.

Exploit Scenario
Eve, a malicious actor, notices that the `initialize` function can be called directly on the contract. She calls it and makes herself an owner while advertising that the contract is actually legitimate. As a result, users may believe Eve as the contract was originally deployed by the Offchain Labs team.

Recommendations
Short term, add the `_disableInitializers` function to the constructor.

Long term, thoroughly document the risks of leaving initializers unprotected and review the codebase looking for any other contracts that may not have their initialize functions protected.

3. Failing cohort elections can block future elections

Severity: **Medium**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-ASC-3
Target: `src/security-council-mgmt/governors/SecurityCouncilNomineeElectionGovernor.sol`	

Description

The introduction of the requirement that the last election has to be executed to be able to create the next election can put the election system in a denial-of-service (DoS) state.

The `createElection` function in the `SecurityCouncilNomineeElectionGovernor` contract validates whether the previous member election has been executed; if it has not, it reverts.

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

Figure 3.1: The `createElection` function in SecurityCouncilNomineeElectionGovernor.sol#L159-L161

However, as shown in the figure below, there are some cases in which an election can become “unexecutable” and therefore block any future elections from being created, essentially preventing the council from being reelected.

```
function _execute(
    uint256 proposalId,
    address[] memory, /* targets */
    uint256[] memory, /* values */
    bytes[] memory callDatas,
    bytes32 /*descriptionHash*/
) internal virtual override {
    // we can only execute when the vetting deadline has passed
    uint256 vettingDeadline = proposalVettingDeadline(proposalId);
    if (block.number <= vettingDeadline) {
        revert ProposalInVettingPeriod(block.number, vettingDeadline);
    }
```

```
    uint256 cnCount = compliantNomineeCount(proposalId);
    uint256 cohortSize = securityCouncilManager.cohortSize();
    if (cnCount < cohortSize) {
        revert InsufficientCompliantNomineeCount(cnCount, cohortSize);
    }

    [...]
}
```

Figure 3.2: The `_execute` function in SecurityCouncilNomineeElectionGovernor.sol#L319-L346

An example is when the number of nominees is below the expected cohort size (currently, that means less than six nominees); it is impossible to recover from this situation without a contract upgrade because there is no way to add more nominees (through the `includeNominee` function) after the vetting period has ended.

Note that this is not the only way an election may become "blocked," as the security council could technically make changes to the `SecurityCouncilManager` that would prevent execution, or the Foundation could exclude a nominee from the list and force the situation outlined above to occur.

Exploit Scenario

A new election starts for which there are six candidates; however, one of the candidates does not meet the requirements of the Foundation and therefore is excluded, leaving only five candidates.

The vetting period ends with no additional nominee being proposed, which prevents the election from being executed (as there are not enough nominees for the cohort) and also prevents any future elections from being created.

Recommendations

Short term, consider allowing the nominee vetter to include nominees after the vetting period only if there are insufficient nominees to meet the cohort size requirement.

Long term, thoroughly document the expected behavior of the elections (throughout their lifecycle). Determine whether there are cases that may block future elections from occurring and how to recover from those situations.

4. Project dependencies contain vulnerabilities	
Severity: **Informational**	Difficulty: **High**
Type: Patching	Finding ID: TOB-ASC-4
Target: All contracts inheriting from the Governor	

Description

Although dependency scans did not yield a direct threat to the project under review, `npm` and `yarn audit` identified dependencies with known vulnerabilities. In particular, the use of OpenZeppelin's contracts of version 4.9.1 or less could be affected by the following front-running vulnerability (CVE-2023-34234).

```
By frontrunning the creation of a proposal, an attacker can become the proposer and
gain the ability to cancel it. The attacker can do this repeatedly to try to prevent
a proposal from being proposed at all.

This impacts the Governor contract in v4.9.0 only, and the
GovernorCompatibilityBravo contract since v4.3.0.
```

Figure 4.1: OpenZeppelin contracts' governor proposal creation may be blocked by front-running

The issue by default affects only version 4.9.0 because it externally exposes the `cancel` function. However, the current codebase uses 4.7.3, in which the `cancel` function is not exposed; the CVE is therefore not exploitable.

Recommendations

Short term, update build process dependencies to the latest versions wherever possible. Use a tool such as `yarn audit` to confirm that no vulnerable dependencies remain in the codebase. Additionally, document this risk so that future contract upgrades do not make it exploitable.

Long term, implement dependency checks as part of the CI/CD pipeline. Do not allow builds to continue with vulnerable dependencies.

5. Security council member can be removed unexpectedly

Severity: **Low**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-ASC-5
Target: `src/security-council-mgmt/governors/SecurityCouncilMemberRemovalGovernor.sol`	

Description

A member can be removed either by the Security Council or by the DAO after an election. However, since the DAO election is not tied to the current Security Council, it is possible to use an unexecuted proposal started for the Security Council N for the Security Council N+1.

The `propose` function to remove a member correctly checks that the address to remove is effectively a member of the Security Council.

```
function propose(
     address[] memory targets,
     uint256[] memory values,
     bytes[] memory calldatas,
     string memory description
 ) public override returns (uint256) {
     ...
     address memberToRemove = abi.decode(rest, (address));
     if (
         !securityCouncilManager.firstCohortIncludes(memberToRemove)
             && !securityCouncilManager.secondCohortIncludes(memberToRemove)
     ) {
         revert MemberNotFound(memberToRemove);
     }
     ...
 }
```

Figure 5.1: Part of the `propose` function in `SecurityCouncilMemberRemovalGovernor.sol#L103-L140`

However, in some specific cases, it is possible that even if the proposal succeeds, its execution will not. For example, if the proposal is started just before a member election is executed, it will effectively replace the member that the removal proposal wants to remove. In that case, when executing the removal proposal, it will revert because the

member is no longer part of the Security Council. Note that the proposal execution does not have any timeline, so it is always valid.

Exploit Scenario

A removal proposal for member A is created at time N. At time N+1, a member election is executed, and member A is replaced by a new member. At the next member election, member A is re-elected; however, anyone can execute the removal proposal created at time N to remove member A unexpectedly.

Recommendations

Short term, consider adding a deadline for the creation of a removal proposal (e.g., two weeks before the next member election starts). Note that this does not guarantee that the proposal, if successful, will be executed. Another solution could be to tie a removal proposal to the current Security Council state.

Long term, document the expected behavior when the timeline of actions overlaps.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Quality Recommendations

- The `blockNumber > endBlock` condition can be updated to `blockNumber >= endBlock` because if `blockNumber` is equal to `endBlock`, the votes will be 0 due to the linearly decreasing weight system.

```
// After proposalDeadline all votes have zero weight
 uint256 endBlock = proposalDeadline(proposalId);
 if (blockNumber > endBlock) {
     return 0;
 }
```

Figure B.1: Code snippet of the `votesToWeight` function (SecurityCouncilMemberElectionGovernorCountingUpgradeable.sol#L234-L238)

- The `InvalidCohort` error is never used.

```
error InvalidCohort(Cohort cohort);
```

Figure B.2: `InvalidCohort` error (ISecurityCouncilManager.sol#L34)

- There is a typo in `UpgadeExecDoesntExist`. This error should be `UpgradeExecDoesntExist`.

```
error UpgadeExecDoesntExist(uint256 chainId);
```

Figure B.3: UpgadeExecDoesntExist error (UpgradeExecRouteBuilder.sol#L40)

C. Fuzzing

Trail of Bits developed differential fuzzing between Solady's `insertionSort` function and an unoptimized version of the insertion sort algorithm to find possible mistakes. However, we found no divergent behavior. To run it with Echidna or Medusa, use the following command (make sure to use assertion mode):

- `echidna solady_diff.sol --contract Test --test-mode assertion`.
- `medusa fuzz solady_diff.sol --deployment-order Test`.

The figure below contains the code we used to perform the differential for Solady's `insertionSort` function.

```
pragma solidity 0.8.19;

contract Test {
    uint256[] data;

    function add_element(uint256 elem) external {
      data.push(elem);
    }

    function testInsertionSort() external {
      uint256[] memory data_res = new uint256[](data.length);
      bool reverted;

      try this.insertionSort(data) returns(uint256[] memory data){
        for(uint256 i; i < data.length; i++) {
          data_res[i] = data[i];
        }
        reverted = false;
      } catch(bytes memory) {
        reverted = true;
      }

      try this.soladyInsertionSort(data) returns(uint256[] memory data_opt){
        assert(!reverted);
        for(uint i; i < data.length; i++) {
          assert(data_res[i] == data_opt[i]);
        }
      } catch(bytes memory) {
        assert(reverted);
      }
    }

    function insertionSort(uint256[] memory data) external returns(uint256[]
memory){
      for (uint i = 1; i < data.length; i++) {
          uint key = data[i];
```

```
            int j = int(i) - 1;
            while ((int(j) >= 0) && (data[uint(j)] > key)) {
                data[uint(j + 1)] = data[uint(j)];
                j--;
            }
            data[uint(j + 1)] = key;
        }
        return data;
    }

    function soladyInsertionSort(uint256[] memory a) external returns(uint256[]
memory){
        /// @solidity memory-safe-assembly
        assembly {
            let n := mload(a) // Length of `a`.
            mstore(a, 0) // For insertion sort's inner loop to terminate.
            let h := add(a, shl(5, n)) // High slot.
            let s := 0x20
            let w := not(0x1f)
            for { let i := add(a, s) } 1 {} {
                i := add(i, s)
                if gt(i, h) { break }
                let k := mload(i) // Key.
                let j := add(i, w) // The slot before the current slot.
                let v := mload(j) // The value of `j`.
                if iszero(gt(v, k)) { continue }
                for {} 1 {} {
                    mstore(add(j, s), v)
                    j := add(j, w) // `sub(j, 0x20)`.
                    v := mload(j)
                    if iszero(gt(v, k)) { break }
                }
                mstore(add(j, s), k)
            }
            mstore(a, n) // Restore the length of `a`.
        }
        return a;
    }

}
```

Figure C.1: The code for the differential fuzzing of Solady's `insertionSort` function

We also implemented differential fuzzing between Solady's `DateTimeLib` and the `electionTimestamp` function, which calculates the timestamp of the next election. In this case, we observed some small discrepancies (in the magnitude of one to three days); however, we believe that these stem from the fact that the `electionTimestamp` function manipulates only the year and month of the first nomination start date and then converts it to a timestamp using Solady's `dateTimeToTimestamp`. By contrast, in our case, we convert the original date to a timestamp and use Solady's `addMonths` function to calculate the timestamp of the next election, which accounts for leap years.

Although these errors can be acceptable, we recommend the Offchain Labs team to take a closer look at these tests and consider whether the difference is acceptable.

The figure below contains the code we used to perform the differential between the `electionTimestamp` function and a version using Solady's `DateTimeLib` functions.

```
pragma solidity 0.8.16;

import "./DateTimeLib.sol";

contract Test2 {

    struct Date {
        uint256 year;
        uint256 month;
        uint256 day;
        uint256 hour;
    }

    Date public _firstNominationStartDate;
    bool public _dateIsSet;

    event Election(uint256);

    function setDate(uint256 day, uint256 month, uint256 year) public {
        if (day == 0 || day > 31) return;
        _firstNominationStartDate.day = day;

        if (month < 1 || month > 12) return;
        _firstNominationStartDate.month = month;

        if (year < 1970 || year >= 0xffffffff) return;
        _firstNominationStartDate.year = year;

        _firstNominationStartDate.hour = 0;

        _dateIsSet = true;

        // make sure we are putting a valid date time
        if (!DateTimeLib.isSupportedDateTime(year, month, day, 0, 0, 0)) return;
    }

    function checkElectionAlwaysSixMonthsInTheFuture(uint256 index) public {
        if (!_dateIsSet) return;

        // this is not necessary
        if (index > 10000) return;

        // calculate next election
        uint256 nextElection = electionToTimestamp(index);

        // next election should happen at StartTimestamp + 6 * index
```

```
        uint256 expectedElection = DateTimeLib.addMonths(
            DateTimeLib.dateTimeToTimestamp(
                _firstNominationStartDate.year,
                _firstNominationStartDate.month,
                _firstNominationStartDate.day,
                0,
                0,
                0
            ),
            6 * index
        );


        emit Election(nextElection);
        emit Election(expectedElection);
        assert(nextElection == expectedElection);
    }

    function electionToTimestamp(uint256 electionIndex) internal 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
    });
    }
}
```

Figure C.2: The code for the differential fuzzing of the `electionTimestamp` function