Code 423n4

ElasticDAO Contest

Findings & Analysis Report

March 20, 2021

Overview

About C4

Code 432n4 (C4) is an open organization that consists of security researchers, auditors, developers, and individuals with domain expertise in the area of smart contracts.

A C4 code contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the code contest outlined in this document, C4 conducted an analysis of ElasticDAO's smart contract system written in Solidity. The code contest took place between February 25, 2021 and March 3, 2021.

Wardens

9 Wardens contributed reports to the ElasticDAO code contest:

- Christoph Michel
- Gerard Persoon
- Janbro (Alejandro Muñoz-McDonald)
- Noah Citron
- Paulius
- s1m0
- PocoTiempo (Team)
    - Rajeev
    - Mariano Conti
    - Maurelian

Judge

This contest was judged by Zak Cole.

Final report assembled by Zak Cole, John Patten, Nathaniel Fried, and Adam Avenir.

Summary

The C4 analysis yielded an aggregated total of 27 unique vulnerabilities.

Of these vulnerabilities, 7 received a risk rating in the category of **HIGH** severity, 7 received a risk rating in the category of **MEDIUM** severity, and 10 received a risk rating in the category of **LOW** severity.

C4 analysis also identified an aggregate total of 2 **non-critical recommendations** and 8 **gas optimizations**.

The ElasticDAO team responded to the issues identified as result of this code contest and provided information regarding any changes to the codebase with a pull request. Links to the aforementioned PRs are appended to the issue descriptions outlined within the corresponding details described in the Issues Found By Severity section of this document. A small set of vulnerabilities and submissions were disputed by the ElasticDAO team. For each of these issues, a supporting explanation for these disputes is detailed in the Disputed Findings section.

Scope

Code

The code under review can be found within the C4 ElasticDAO code contest repository and comprises 2,220 lines of code across a total of 13 smart contracts written in the Solidity programming language.

File	Lines of Code
ElasticDAO.sol	**466**
ElasticDAOFactory.sol	**209**
IElasticToken.sol	**49**
ElasticMath.sol	**161**
SafeMath.sol	**94**
DAO.sol	**105**
Ecosystem.sol	**114**
EternalModel.sol	**107**
Token.sol	**104**
TokenHolder.sol	**65**
Configurator.sol	**128**
ReentryProtection.sol	**26**
ElasticGovernanceToken.sol	**592**

This code, including tests and tooling, is also available at the following URL:
https://github.com/elasticdao/contracts/tree/c657b84469ba33efd8914c7e847830d82cb0f3ca

System Overview

Elastic DAO is a governance protocol that attempts to balance the competing interests between the different participants in a decentralized ecosystem. Elastic DAO achieves this by reducing the overall influence that money and early adopters have in existing DAO governance models.

At the time of writing, voting within the ElasticDAO system relies on the Snapshot platform with an anticipated implementation of an independent layer 2 solution following their launch. With this in mind, a majority of the voting logic is executed through a multisig. This multisig also acts as administrator within the context of the proxy, ElasticDAO controller, burner, and minter contracts.

Further documentation can be found here.

Contract Logic

`core/ElasticDAO.sol` defines the logic for deploying, initializing, summoning, joining, and exiting a DAO.

`core/ElasticDAOFactory.sol` provides a singular approach for deploying DAOs and is meant to be managed by the first DAO, ElasticDAO.

`tokens/ElasticGovernanceToken.sol` is a rebasing token that conforms to the ERC20 spec.

Storage contracts follow a version of the Eternal Storage pattern and are found in `src/models`.

As this code conforms to NatSpec formatting specifications, lower level details regarding function can be found as comments within the code itself.

Severity Criteria

C4 assesses severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into 3 primary risk categories:

- Low (1)
- Medium (2)
- High (3)

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

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

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided in the C4 GitHub repository.

Issues Found By Severity

High Severity

[H-01] Infinite minting of tokens by exploiting eternal storage pattern on DAO.sol
Attackers can overwrite metadata in the `models/DAO.sol` eternal storage contract by using the `serialize` call to change the expected configurator address to the attacker's address.

This allows the attacker to change the DAO data and potentially mint infinite tokens for themselves. We consider this risk high severity because it would disrupt the economics of the DAO in a manner that would prevent the system from performing.

> *ElasticDAO: Confirmed and resolved in PR #43.*

[H-02] Infinite minting of tokens by exploiting eternal storage pattern on Ecosystem.sol
A similar exploit is possible because of the `models/Ecosystem.sol` eternal storage contract. Attackers can change the expected address to their own address while bypassing the authorization check for this function. This allows the attacker to change the DAO ecosystem data.

Gaining full access to the DAO ecosystem data is also possible by changing the `daoModelAddress` field to an attacker-controlled proxy contract. This attack vector allows attackers to mint infinite tokens for themselves and potentially break all DAOs created by the ElasticDAO system.

> *ElasticDAO: Confirmed and resolved in PR #43*

[H-03] Infinite minting of tokens by exploiting eternal storage pattern on Token.sol
An attacker can change the expected DAO address to their own address in the `models/Token.sol` contract. The attacker can change the token parameters so that they receive much more ETH for their shares when exiting from the DAO. Attackers could also steal all funds from the DAO, effectively breaking ElasticDAO's model.

> *ElasticDAO: Confirmed and resolved in PR #43*

[H-04] Configurator contract allows infinite minting of tokens
Even if a configurator address is hard-coded in the `Ecosystem.sol` contract, attackers can call the `buildToken` function on the `services/Configurator.sol` contract and change nearly all of the parameters to be attacker-controlled. The attacker bypasses the authorization check and gains write access to fields of the DAO ecosystem.

ElasticDAO: Confirmed and resolved in PR #43

[H-05] New users can be blocked from joining
The `join` function requires users to send an exact amount of ETH for the required shares. Attackers can survey the mempool for `join` transactions and send a tiny amount of wei to the DAO contract to change the required ETH value to a different value than the user submitted. This griefing attack would prevent new users from joining the DAO.

Malicious actors might use this attack to prevent new votes from derailing the outcome of a proposal. Normal DAO usage might also block new users from joining. Token curve parameters change as a result of new join transactions. During times of high demand, many transactions will fail and effectively result in a self-DOS.

ElasticDAO: Confirmed and resolved in PR #43 and PR #59

[H-06] Incorrect event parameters in `transferFrom` function
The emitApproval event should occur when the msg.sender is not equal to _from. The event should be emit Approval(from, msg.sender, _allowances[_from][msg.sender]); instead of emit Approval(msg.sender, _to, _allowances[_from][msg.sender]). This error may negatively impact off-chain tools that are monitoring critical transfer events of the governance token.

ElasticDAO: Confirmed and resolved in PR #77

[H-07] Minter can call functions reserved for DAO addresses
In `ElasticGovernanceToken.sol`, the `onlyDAO` modifier is meant to only allow a DAO address to call functions like `setBurner` and `setMinter`. However, as currently written, this modifier allows the msg.sender to either be a DAO address or the minter address.

ElasticDAO: Confirmed and resolved in PR #54

Medium Severity

[M-01] No check to prevent fee burning

The `collectFees` function sends fees to a `feeAddress` in storage, however, there is currently no check to validate whether or not `feeAddress` has been initialized. An attacker can call `collectFees` to send the fees to the zero address, making recovery impossible. This attack could be used against new DAOs to burn their main revenue besides the token market.

> *ElasticDAO: Confirmed and resolved in PR #42*

[M-02] Anyone can update the number of token holders

The `updateNumberOfTokenHolders` function in the `ElasticGovernanceToken.sol` does not verify the caller. Anyone could call this function and set the value to `0`. While this does not put funds at risk, an attacker could set the value of `numberOfTokenHolders` to `MAX_UINT`, resulting in an overflow and a reverted transaction the next time `updateNumberOfTokenHolders` is called to to increment this number.

> *ElasticDAO: Confirmed and resolved in PR #53*

[M-03] The `initialize` function does not check for non-zero values

The `initialize` function does not check if the summoners are all non-zero addresses. If all the initialized summoners happen to be 0, the contract will have to be redeployed.

> *ElasticDAO: Confirmed and resolved in PR #78*

[M-04] Potential for lock out of administrative access

The `setController` function in ElasticDAO.sol updates the controller address in one set-up. If the controller address is set incorrectly, administrative access is prevented because `setController` includes an onlyController modifier. The contract would have to be redeployed if this mistake is made.

> *ElasticDAO: "The vulnerability is correct, however, the impact is incorrect. Because we deploy with proxies, in a worst case scenario, the proxy implementation could be upgraded to fix this issue."*

[M-05] Malicious actors can avoid penalty
A DAO member may be able to predict when they will be penalized if they monitor the mempool for events related to the `penalize` function on the contract. This member can then avoid penalization by transferring their balance to another address and sending it back to the original account after the next block.

Since the penalty transaction will revert if the amount is greater than the balance, an attacker could potentially frontrun the penalty by calling the `exit` function with a miniscule amount of ETH. They could also exit the DAO completely. This loophole provides potential incentive for malicious actors to exploit the DAO.

> *ElasticDAO: Confirmed and resolved in PR #44*

[M-06] Double-spend allowance
A malicious attacker can execute a double-spend attack on an allowance by front-running the execution of an `approve()` function that alters the state of a balance. Since the `increaseAllowance` and `decreaseAllowance` functions provide the same functionality, the `approve()` function is an unnecessary attack vector that can present significant risk.

> *ElasticDAO: "This issue is present in most ERC20 tokens and very few choose to take the recommended mitigation step. We've chosen to go with expected behaviour instead of removing a function that is part of the spec."*

[M-07] Passing a zero address for controller will require redeployment of the contract
Passing a zero address for the controller during initialization will require redeployment of the contract because the `onlyController` modifier for critical contract functions cannot be changed after initial deployment.

> *ElasticDAO: Confirmed and resolved in PR #47*

Low Severity

[L-01] Wrong logic check in `initialize` function
The `initialize` function has a check that passes when either the `_ecosystemModelAddress` or the `_controller` address are non-zero values. Unless the function checks whether both are non-zero, DAOs could be deployed with incorrect parameters (e.g., the controller set as the zero address).

If the controller address is the zero address, then functions with the `onlyController` modifier would be unusable – specifically, `penalize`, `reward`, and `setController` functions. There would be no way to change the parameters at a later time and the DAO would have to be redeployed to address this issue.

> *ElasticDAO: Confirmed and resolved in PR #47*

[L-02] Malicious summoner could prevent entry of new DAO members
The `summon` function allows a summoner to define the initial shares that each summoner receives. A malicious summoner can mint the total number of summoner shares to be equal to the maximum possible value of an unsigned integer. Any attempt to mint new shares beyond this initial supply would fail and result in an overflow. This would prohibit the ability for new members to join the DAO, which would need to be redeployed.

> *ElasticDAO: "Summoners are considered to be coordinating, trusted entities. We do not consider it to be a bug that needs fixing, despite being technically accurate."*

[L-03] The `summon` function can be called prior to all deposits being received
A summoner can prevent others from receiving initial shares by prematurely calling the `summon` function. This function can be called as soon as the DAO contract receives a non-zero amount of ETH. A malicious summoner could be the first to seed ETH to the contract then prevent anyone else from joining. Summoners who were willing to seed the DAO would only be able to receive their shares through a reward transaction sent by the multisig account or by redeploying the DAO.

> *ElasticDAO: "Summoners are considered to be coordinating, trusted entities. We do not consider it to be a bug that needs fixing, despite being technically accurate."*

[L-04] Potential underflow caused by DAO exit
`Safemath` is not utilized consistently in the `ElasticDAO.sol` contract, potentially causing underflow when DAO members exit. They may leave an amount of ETH smaller than the amount purchased. Underflow could result in a large number of tokens being minted by `msg.sender`.

> *ElasticDAO: Confirmed and resolved in PR #48*

[L-05] Incomplete `serialize` function
The `deserialize` function can deserialize more fields than can be serialized with the `serialize` function. Specifically, the `numberOfTokenHolders` field can only be serialized with the `updateNumberOfTokenHolders` function. This does not result in any significant security vulnerabilities, but can cause problems in the future composability.

> *ElasticDAO: "We do not view this as an issue. It may be considered by some to be bad practice, but it improves gas efficiency in our case."*

[L-06] Missing call to Safemath
The `join` function includes a subtraction that is made without a call to `Safemath`. Since this function involves ETH, an errant value could be problematic.

> *ElasticDAO: Confirmed and resolved in PR #48*

[L-07] Missing calls to `Safemath`
The `wdiv` function involves two divisions used without a `Safemath.div` call.

> *ElasticDAO: Confirmed and resolved in PR #46*

[L-08] decreaseAllowance must be greater than 0
The `decreaseAllowance` function requires that the new allowance must be greater than 0. The `require` argument should allow users to decrease the allowance to zero.

> *ElasticDAO: Confirmed and removed entirely in PR #81*

[L-09] Max voting limitation can be manipulated via Sybil attack
ElasticDAO implements a join curve to make Sybil attacks prohibitively expensive since users cannot purchase more than the DAO-configured maximum number of tokens. As more addresses join, the more expensive additional attacks become.

We found that attackers could circumvent this restriction if a DAO member has more than the max number of tokens permitted for voting. This member could join the DAO with another address using a negligible amount of ETH and transfer shares from their primary account. This attack vector breaks the current voting model, thereby rendering the max voting token restriction ineffective.

> *ElasticDAO: Disputed, partially resolved in PR #59*
>
> *ElasticDAO maintains that this is a strength of the protocol rather than a weakness. To successfully sybil attack the network, the attacker would need to purchase so many tokens on the open market that it would drastically inflate the value of existing members' shares. These members could also exit from the DAO and start a new one. The financial incentives in ElasticDAO serve as a protection against attacks such as these. It would be counterproductive and unfeasible to attack DAOs in this way.*

[L-10] Excessively strict `penalize` function can result in reverted transactions
When calling the penalize function, unless the amount is less than or equal to the available lambda, the transaction will revert due to `SafeMath.sub` failing to execute `updateBalance`.

> *ElasticDAO: Confirmed and resolved in PR #44*

Non-Critical Risks

[N-01] Inconsistent values for the transfer event

The `burnShares` function emits a transfer event passing `_deltaLambda` as the amount transferred. The `mintShares` function uses `deltaT`. While this does not create a security risk, it may make it harder for a frontend application to handle the values coming from these events.

> *ElasticDAO: Confirmed and resolved in PR #52*

[N-02] For loop in serial function

The `serialize` function sets summoners using a `for` loop. If the list of summoners is updated and the new list is smaller than the first, the remaining summoners are still considered active. The loop would include the excess elements. We judge this to be a non-critical risk as the function can only be called as a trusted party, and the list of summoners is unlikely to be updated.

> *ElasticDAO: "This should not be an issue, as summoners are only set before the DAO is summoned. Additionally, the summoners have no special case or reason for existence after summoning."*

Gas Optimizations

[O-01] Eliminate repetition of deployer check
The `initializeToken` function validates that the value of `msg.sender` is equal to that of the deployer. Since this check is already executed in the `onlyDeployer` modifier, it should be considered redundant in order to reduce gas consumption.

> *ElasticDAO: Confirmed and resolved in PR #56*

[O-02] Unused arguments
The second argument in `models/Dao.exists` is unused.

> *ElasticDAO: Confirmed and resolved in PR #57*

[O-03] Inefficient calculations
`SafeMath.pow` is less efficient than a repeated squaring algorithm. Furthermore, the exponentiation in `ElasticMath.revamp` is very expensive and can be hardcoded as `1e18`.

> *ElasticDAO: Confirmed and resolved in PR #58*

[O-04] Unnecessary storage of summoners
`ElasticDao.sol` stores the summoners in its storage, despite only using the summoners stored in the DAO eternal storage model. To save gas, the summoners storage within ElasticDAO can be removed.

> *ElasticDAO: Acknowledged, This is for convenience on the frontend. The one time gas cost replaces repeated O(n) calls to the node with a single call (O(1)).*

[O-05] Unnecessary use of preventReentry modifiers
Many uses of the `preventReentry` modifiers are used on functions that call trusted contracts and do not take any attacker-controlled arguments. These include: `ElasticDAO.initialize`, `initializeToken`, `exit`, `join`, `penalize`, `reward`, `setController`, `setMaxVotingLambda`, `seedSummoning`, `summon`, `ElasticDAOFactory.initialize`, `collectFees`, `deployDAOAndToken`, `updateElasticDAOImplementationAddress`, `updateFee`, `updateFeeAddress`, `updateManager`, and several `ElasticGovernanceToken` functions.

> *ElasticDAO: Confirmed and resolved in PR #79*

[O-06] Unnecessary calculation of deployedDAOCount
The `ElasticDAOFactory` tracks `deployedDAOCount` in a separate variable. The `deployDAOAndToken` function updates this value. This is an unnecessary calculation and invocation of `Safemath` because the `deployedDAOCount` can be returned with a view that returns `deployedDAOAddresses.length`.

> *ElasticDAO: Confirmed and resolved in PR #51*

[O-07] Unnecessary call of ElasticMath
The `ElasticGovernanceToken` and `ElasticDAO` contracts import both `Safemath` and `ElasticMath`. `ElasticMath` already imports `Safemath`.

> *ElasticDAO: Confirmed and resolved in PR #50*

[O-07] Unnecessary checks in `setBurner` and `setMinter` functions
These functions check for return values, which is unnecessary. The Boolean will always return true if complete without reverting.

> *ElasticDAO: "We like the additional safety of the checks. Gas costs are less important as this function is called infrequently."*

[O-08] Unnecessary use of `bytes32` setting Name

The events `MaxVotingLambdaChanged` and `ControllerChanged` have this parameter, which serves no purpose because the only place these events are emitted are in the functions `setMaxVotingLambda` and `setController`.

ElasticDAO: Confirmed and resolved in PR #85

Disputed Findings

[D-01] Multi-signature threshold not specific in contracts

ElasticDAO's documentation states that the DAO's controller is a multi-signature account with nine members. However, the code makes no mention of this stipulation, nor does it reference the concept of a multi-signature contract. The actual functionality of ElasticDAO might be different than what the documentation suggests, which could potentially result in confusion or financial loss for misinformed investors.

> *ElasticDAO disputed this bug, saying: "The documentation is a set of living documents. They, like all project documentation, are subject to change. The controller address in ElasticDAO.sol is designed to be a multisig. We currently envision that to be a 9 member multisig, but that may change leading into launch, at which point the documentation will be updated. Should there ever be a discrepancy between the documentation and the actual address stored as the controller, an investor only has to look at the contract on etherscan to see that we are tricking possible investors. Looking at the code on etherscan would be the only way to verify that the contracts themselves enforced this 9 member requirement, so looking at the code of the controller address is not any more difficult. Should this controller ever be changed, the ControllerChanged event would be fired, providing investors with the chance to actively monitor changes and notice any trickery. All of this also ignores the long term financial disincentive involved in tricking potential investors. Even if the approach described above is not trustless for the warden, the scope of this contest was the code, not the documentation."*

[D-02] Allowances mapping does not implement eternal storage pattern

Most of the data is stored using the eternal storage pattern, but the mapping `_allowance` does not. This could result in unexpected problems after upgrades, but we assess this as a low probability risk.

> *ElasticDAO disputed this bug, saying: "This choice was made primarily for gas reasons. The worst case scenario, a full loss of allowance data, is that every wallet needs to re-approve the spending of their tokens."*

[D-03] `seedSummoning` **mints the incorrect number of shares**

Since the amount of eth is being divided by eth per share, the incorrect number of shares are being minted. Improper minting could result in system failure.

> *ElasticDAO disputed this bug, saying: "The referenced functionality is performing as expected. It's possible that the warden did not understand intent, but the report is incorrect."*

[D-04] `seedSummoning` **mints more tokens than expected**

The `wdiv` function converts the input amount to wei by multiplying by `10^8`. However, the input amount is already expressed in `wei`. This causes `10^18` more tokens to be minted as expected when a summoner calls the `seedSummoning` function.

> *ElasticDAO disputed that summoners can mint `10^18` more tokens than expected. This bug was not reproducible.*

[D-05] ElasticMath functions could result in 0

The function `wmul` in `ElasticMath.sol` can produce unexpected results if the operands are positive integers. Multiplication of two positive numbers `a` and `b` can result in 0 for every `a, b > 0` and `b < 5*10^7 / a`. This could result in a critical error since `wmul` is used for all arithmetic functions on tokens.

> *ElasticDAO disputed this bug. The team engaged with several outside mathematics experts and the original developers of the function. These experts agreed that the resulting value produced by wmul is less than the minimum supported value in Solidity and does not present any risk. A more concise explanation of their response and findings are outlined in a public Gist document.*

Disclosures

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

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provide 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.