

Report

v. 1.0

Customer

Eco



Smart Contract Audit

Eco. Phase II

5th November 2022

# Contents

<b>1 Changelog</b>	<b>6</b>
<b>2 Introduction</b>	<b>7</b>
<b>3 Project scope</b>	<b>8</b>
<b>4 Methodology</b>	<b>10</b>
<b>5 Our findings</b>	<b>11</b>
<b>6 Critical Issues</b>	<b>12</b>
CVF-2.1. FIXED	12
<b>7 Major Issues</b>	<b>13</b>
CVF-2.2. FIXED	13
CVF-2.3. FIXED	13
CVF-2.4. FIXED	13
CVF-2.5. FIXED	14
CVF-2.6. FIXED	14
CVF-2.7. FIXED	14
CVF-2.8. FIXED	15
CVF-2.9. FIXED	15
CVF-2.10. FIXED	15
CVF-2.11. FIXED	16
CVF-2.12. FIXED	16
CVF-2.13. FIXED	16
CVF-2.14. FIXED	17
CVF-2.15. FIXED	17
CVF-2.16. FIXED	17
CVF-2.17. FIXED	18
CVF-2.18. FIXED	18
CVF-2.19. FIXED	19
CVF-2.20. FIXED	19
<b>8 Moderate Issues</b>	<b>20</b>
CVF-2.21. FIXED	20
CVF-2.22. INFO	20
CVF-2.23. FIXED	21
CVF-2.24. FIXED	21
CVF-2.25. FIXED	21
CVF-2.26. FIXED	22
CVF-2.27. FIXED	22
CVF-2.28. FIXED	22
CVF-2.29. FIXED	23

CVF-2.30. FIXED	23
CVF-2.31. FIXED	23
CVF-2.32. FIXED	24
CVF-2.33. INFO	24
CVF-2.34. INFO	25
CVF-2.35. INFO	25
<b>9 Minor Issues</b>	<b>26</b>
CVF-2.36. FIXED	26
CVF-2.37. FIXED	26
CVF-2.38. FIXED	26
CVF-2.39. FIXED	27
CVF-2.40. FIXED	27
CVF-2.41. INFO	27
CVF-2.42. FIXED	27
CVF-2.43. FIXED	28
CVF-2.44. FIXED	28
CVF-2.45. FIXED	28
CVF-2.46. INFO	29
CVF-2.47. FIXED	29
CVF-2.48. FIXED	29
CVF-2.49. FIXED	29
CVF-2.50. FIXED	30
CVF-2.51. INFO	30
CVF-2.52. FIXED	30
CVF-2.53. FIXED	30
CVF-2.54. FIXED	31
CVF-2.55. FIXED	31
CVF-2.56. FIXED	31
CVF-2.57. FIXED	31
CVF-2.58. FIXED	32
CVF-2.59. INFO	32
CVF-2.60. INFO	33
CVF-2.61. FIXED	33
CVF-2.62. FIXED	33
CVF-2.63. FIXED	34
CVF-2.64. FIXED	34
CVF-2.65. FIXED	35
CVF-2.66. FIXED	35
CVF-2.67. INFO	36
CVF-2.68. FIXED	36
CVF-2.69. FIXED	36
CVF-2.70. FIXED	37
CVF-2.71. FIXED	37
CVF-2.72. FIXED	37
CVF-2.73. FIXED	38

CVF-2.74. FIXED	38
CVF-2.75. FIXED	38
CVF-2.76. FIXED	39
CVF-2.77. FIXED	39
CVF-2.78. FIXED	39
CVF-2.79. FIXED	40
CVF-2.80. FIXED	40
CVF-2.81. FIXED	40
CVF-2.82. FIXED	41
CVF-2.83. FIXED	41
CVF-2.84. FIXED	41
CVF-2.85. FIXED	42
CVF-2.86. FIXED	42
CVF-2.87. FIXED	42
CVF-2.88. FIXED	42
CVF-2.89. FIXED	43
CVF-2.90. FIXED	43
CVF-2.91. FIXED	43
CVF-2.92. FIXED	43
CVF-2.93. FIXED	44
CVF-2.94. FIXED	44
CVF-2.95. FIXED	44
CVF-2.96. FIXED	44
CVF-2.97. FIXED	45
CVF-2.98. FIXED	45
CVF-2.99. FIXED	45
CVF-2.100. FIXED	46
CVF-2.101. FIXED	46
CVF-2.102. FIXED	46
CVF-2.103. FIXED	47
CVF-2.104. FIXED	47
CVF-2.105. INFO	47
CVF-2.106. FIXED	48
CVF-2.107. FIXED	48
CVF-2.108. FIXED	48
CVF-2.109. FIXED	48
CVF-2.110. FIXED	49
CVF-2.111. FIXED	49
CVF-2.112. FIXED	49
CVF-2.113. FIXED	50
CVF-2.114. FIXED	50
CVF-2.115. INFO	50
CVF-2.116. FIXED	51
CVF-2.117. FIXED	51
CVF-2.118. FIXED	51
CVF-2.119. FIXED	52

CVF-2.120. <b>FIXED</b>	52
CVF-2.121. <b>FIXED</b>	52
CVF-2.122. <b>FIXED</b>	53
CVF-2.123. <b>INFO</b>	54
CVF-2.124. <b>FIXED</b>	55
CVF-2.125. <b>FIXED</b>	55
CVF-2.126. <b>FIXED</b>	55
CVF-2.127. <b>FIXED</b>	56
CVF-2.128. <b>FIXED</b>	56
CVF-2.129. <b>FIXED</b>	56
CVF-2.130. <b>FIXED</b>	57
CVF-2.131. <b>FIXED</b>	57
CVF-2.132. <b>FIXED</b>	57
CVF-2.133. <b>FIXED</b>	58
CVF-2.134. <b>FIXED</b>	58
CVF-2.135. <b>FIXED</b>	58
CVF-2.136. <b>INFO</b>	59
CVF-2.137. <b>INFO</b>	59
CVF-2.138. <b>FIXED</b>	59
CVF-2.139. <b>FIXED</b>	60
CVF-2.140. <b>FIXED</b>	60
CVF-2.141. <b>FIXED</b>	60

# 1 Changelog

#	Date	Author	Description
0.1	04.11.22	A. Zveryanskaya	Initial Draft
0.2	04.11.22	A. Zveryanskaya	Minor revision
1.0	05.11.22	A. Zveryanskaya	Release

## 2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

Eco utilizes community governance and monetary policy to control a digital currency designed to be a decentralized alternative to fiat currency. This audit covers the token contracts as well as the associated subsystem frameworks and governance.

We were asked to review fixes from Phase I and the new functionality in [bece734 commit](#). We checked that the new issues were fixed as of [5733b1d commit](#).



# 3 Project scope

Repositories:

- First Fix Repository
- Second Fix Repository

Files:

## **currency/**

DelegatePermit.sol	ECO.sol	EcoTokenInit.sol
ECOx.sol	EcoXTokenInit.sol	ERC20.sol
ERC20Pausable.sol	ERC20Permit.sol	IECO.sol
InflationCheckpoints.sol	VoteCheckpoints.sol	

## **policy/**

ERC1820Client.sol	Policed.sol	PolicedUtils.sol
Policy.sol	pPolicyInit.sol	

## **governance/**

ECOxLockup.sol	SimplePolicySetter.sol	TimedPolicies.sol
IGeneration.sol	IGenerationIncrease.sol	CurrencyTimer.sol

## **governance/community/**

ECOxStaking.sol	Elect CircuitBreaker.propo.sol	PolicyProposals.sol
VotingPower.sol	PolicyVotes.sol	Trustee Replacement.propo.sol

SingleTrustee  
Replacement.propo.sol

\* *strikethrough files were removed from scope*



**governance/monetary/**

CurrencyGovernance.sol	ILockups.sol	InflationRoot HashProposal.sol
Lockup.sol	RandomInflation.sol	TrustedNodes.sol

**proxy/**

ForwardProxy.sol	ForwardTarget.sol
------------------	-------------------

**utils/**

TimeUtils.sol	StringPacker.sol
---------------	------------------

**VDF/**

BigNumber.sol	IsPrime.sol	VDFVerifier.sol
---------------	-------------	-----------------

Two files were renamed and moved:

- governance/Inflation.sol » governance/monetary/RandomInflation.sol
- governance/ECOxLockup.sol » governance/community/ECOxStaking.sol

Two files were moved:

- governance/Lockup.sol » governance/monetary/Lockup.sol
- governance/ILockups.sol » governance/monetary/ILockups.sol



# 4 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and used technologies, as well as on what the client is expecting from the audit.

- **General Code Assessment.** The code is reviewed for clarity, consistency, style, and for whether it follows code best practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis.** Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places and that their visibility scopes and access levels are relevant. At this phase we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis.** For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and is done properly. At this phase we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis.** The code logic of particular functions is analysed for correctness and efficiency. We check that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

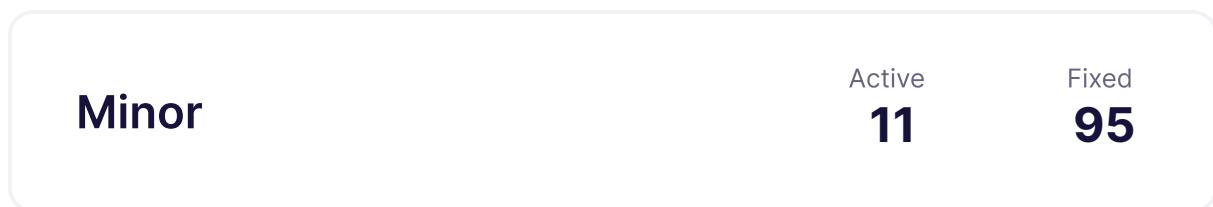
We use next severity levels for found issues:

- **Critical issues** directly affect smart contract functionality and could cause significant losses.
- **Major issues** relate to considerable problems with performance. Also, it could become Critical issue if code modification occurs. In rare cases it relies on unclear code behaviour which should be also double checked..
- **Moderate issues** cannot cause significant losses, but need to be fixed.
- **Minor issues** contain code style, best practices and other recommendations.



# 5 Our findings

We found 1 critical, 19 major, and a few less important issues. All identified Critical and Major issues have been fixed.



Fixed 126 out of 141 issues

# 6 Critical Issues

## CVF-2.1. FIXED

- **Category** Flaw
- **Source** InflationRootHashProposal.sol

**Description** This makes it impossible to challenge a proposal with a zero root hash.

**Recommendation** Consider either forbidding proposing zero root hashes, or allowing challenging them.

**Client Comment** *Added checks to proposeRootHash to make data more predictable.*

1059    +proposal.rootHash != **bytes32(0)**,



# 7 Major Issues

## CVF-2.2. FIXED

- **Category** Unclear behavior
- **Source** ECO.sol

**Description** A separate role for ECO Labs looks weird.

**Recommendation** Consider just assigning a normal role to an address controlled by ECO Labs.

```
73 +msg.sender == policyFor(ID_ECO_LABS) ||
```

## CVF-2.3. FIXED

- **Category** Unclear behavior
- **Source** ECOx.sol

**Description** A separate role for ECO Labs looks weird.

**Recommendation** Consider just assigning a normal role to an address controlled by ECO Labs.

```
221 +msg.sender == policyFor(ID_ECO_LABS),
```

## CVF-2.4. FIXED

- **Category** Unclear behavior
- **Source** EcoXTokenInit.sol

**Description** The returned value is ignored.

**Recommendation** Consider explicitly checking that true was returned.

```
32 +EC0x(_token).transfer(_initialHolders[i], _initialBalances[i]);
```



## CVF-2.5. FIXED

- **Category** Unclear behavior
- **Source** EcoTokenInit.sol

**Description** The returned value is ignored.

**Recommendation** Consider explicitly checking that true was returned.

```
32 +ECO(_token).transfer(_initialHolders[i], _initialBalances[i]);
```

## CVF-2.6. FIXED

- **Category** Documentation
- **Source** VoteCheckpoints.sol

**Description** The comment is irrelevant here, as it actually describes the “getPrimaryDelegate” function.

**Recommendation** Consider rewriting the comment.

```
227 +* @dev Get the primary address `account` is currently delegating to  
+*   ↵ . Defaults to the account address itself if none specified.  
+* The primary delegate is the one that is delegated any new funds  
+*   ↵ the address receives.
```

## CVF-2.7. FIXED

- **Category** Suboptimal
- **Source** RandomInflation.sol

**Description** These checks optimize a very unlikely scenario when a user submits an invalid value, but make a very likely scenario of a valid value more expensive.

**Recommendation** Consider removing these checks.

```
237 +!(_primal % 3 == 0) &&  
+    !(_primal % 5 == 0) &&  
+    !(_primal % 7 == 0) &&  
240 +    !(_primal % 11 == 0) &&  
+    !(_primal % 13 == 0) &&
```



## CVF-2.8. FIXED

- **Category** Suboptimal

- **Source**

InflationRootHashProposal.sol

**Recommendation** It would be cheaper to require “hashes” of the unused subtrees to be just zero, rather than the real root hash of a subtree with zero leafs.

```
977 +if (_proof[i - 1] != ALLOWED_ZERO_DATA[i - 1]) {
```

## CVF-2.9. FIXED

- **Category** Flaw

- **Source** TimedPolicies.sol

**Recommendation** Should be “ $\geq$ ” instead of “ $>$ ”, as “nextGenerationStart” is a valid start time for the next generation.

```
114 +time > nextGenerationStart,
```

## CVF-2.10. FIXED

- **Category** Flaw

- **Source** TimedPolicies.sol

**Description** Here, contract state is changed after calling external contracts, thus reentrancy issues are possible.

**Recommendation** Consider calling external contracts after state updates.

```
158 +startPolicyProposal(mintedOnGenerationIncrease);
```



## CVF-2.11. FIXED

- **Category** Suboptimal
- **Source** TrustedNodes.sol

**Recommendation** This event is already emitted inside the “\_trust” function. Don’t emit again here.

```
112 +emit TrustedNodeAddition(node, cohort);
```

```
310 +emit TrustedNodeAddition(_newCohort[i], cohort);
```

## CVF-2.12. FIXED

- **Category** Procedural
- **Source** TrustedNodes.sol

**Description** Here a series of similar external calls to the same smart contract is performed.

**Recommendation** Consider refactoring to allow fetching all the trusted nodes in one call.

```
137 +for (uint256 i = 0; i < _numTrustees; ++i) {  
+    _trust(TrustedNodes(_self).getTrustedNodeFromCohort(_cohort, i)  
+        );  
+}
```

## CVF-2.13. FIXED

- **Category** Flaw
- **Source** PolicyVotes.sol

**Description** This doesn’t allow changing the amount of “no” votes.

**Recommendation** Consider a situation when both “\_prevVote” and “\_vote” are false, but “\_amount” and “\_oldStake” differ.

**Client Comment** If “\_prevVote” is false, then “\_oldYesVotes” = 0. Therefore the revert is hit only if “\_oldStake” is also 0, and that case is specifically disallowed by the enclosing if statement. However, there is an issue changing the amount of yes votes: \_prevVote = \_vote = true, \_oldStake = \_oldYesVotes != \_amount. Additionally, users weren’t stopped from overwriting their vote with the exact same no vote.

```
144 +_prevVote != _vote || _oldStake != _oldYesVotes,
```



## CVF-2.14. FIXED

- **Category** Suboptimal
- **Source** VDFVerifier.sol

**Description** Strictly speaking, the message doesn't match the condition checked, as the "minimalByteLength" function returns the value rounded up to a whole byte, so a number has to be at least 505 bits long to pass the check.

**Recommendation** Consider rephrasing the message or fixing the check.

```
105 require(  
107 +     y.minimalByteLength() >= MIN_BYTES,  
        "The secret (y) must be at least 512 bit long"  
    );
```

## CVF-2.15. FIXED

- **Category** Suboptimal
- **Source** BigNumber.sol

**Description** This check makes the function less predictable as a number not meant to be aligned could occasionally have factor of 32 bytes.

**Recommendation** Consider implementing two separate functions: one for aligned and another for not aligned numbers.

```
81 require(  
83 +     word != 0,  
        "High-word must be set for 256bit-aligned numbers"  
    );
```

## CVF-2.16. FIXED

- **Category** Documentation
- **Source** BigNumber.sol

**Description** This comment is not relevant anymore.

**Recommendation** Consider removing it.

```
167 * @dev If the caller modifies the returned buffer instance, it will  
    ↪ corrupt the BigNumber value.
```



## CVF-2.17. FIXED

- **Category** Suboptimal

- **Source** BigNumber.sol

**Description** This allocates a new bytes array on every iteration, which is very inefficient.

**Recommendation** Consider allocating a single array of proper size one and then copying all the needed data into it.

**Client Comment** Obseleted by CVF-52.

```
202 +result = bytes.concat(result, bytes32(0x0));
```

```
208 +result = bytes.concat(result, abi.encode(word));
```

```
256 +result = bytes.concat(result, abi.encode(word));
```

## CVF-2.18. FIXED

- **Category** Suboptimal

- **Source** PolicyProposals.sol

**Description** This check was already performed inside the “\_getPaginationBounds” function.

**Recommendation** Consider replacing with: if (\_returnLength == 0) {

**Client Comment** *\_getPaginationBounds is removed.*

```
249 +if (totalProposals == 0 || _startIndex > totalProposals - 1) {
```

```
286 +if (totalProposals == 0 || _startIndex > totalProposals - 1) {
```



## CVF-2.19. FIXED

- **Category** Suboptimal
- **Source** PolicyProposals.sol

**Description** Obtaining the voting power is an expensive operation and should be done as later as possible to save gas.

**Recommendation** Consider placing this line after the double proposal and double stake checks.

```
390 +uint256 _amount = votingPower(msg.sender, blockNumber);
```

## CVF-2.20. FIXED

- **Category** Suboptimal
- **Source** PolicyProposals.sol

**Description** The “totalProposals” variable is read many times.

**Recommendation** Consider reading once and reusing.

```
473 +require(totalProposals > 0, "no proposals to delete");
```

```
477 +uint256 proposalIndex = totalProposals;
+for (uint256 i = 0; i < totalProposals; i++) {
```

```
484 +require(proposalIndex < totalProposals, "proposal does not exist");
```

```
487 +if (proposalIndex < totalProposals - 1) {
+    Proposal lastProposal = allProposals[totalProposals - 1];
```

```
495 +totalProposals = totalProposals - 1;
```



# 8 Moderate Issues

## CVF-2.21. FIXED

- **Category** Unclear behavior
- **Source** InflationCheckpoints.sol

**Description** The returned value is ignored here.

**Recommendation** Consider replacing the “amount” value with the returned value.

```
81 +super._beforeTokenTransfer(from, to, amount);
```

## CVF-2.22. INFO

- **Category** Unclear behavior
- **Source** VoteCheckpoints.sol

**Description** This code undelegates only from the primary delegate, while then tries to delegate the balance, that could include votes still delegated to non-primate delegates.

**Recommendation** Consider delegating only the undelegated amount instead.

**Client Comment** *This is intentional, we have added a comment that this function is not for tandem use with partial delegations and is only to set up PrimaryDelegations which should only have your full balance delegated.*

```
364 +if (!isOwnDelegate(msg.sender)) {  
+    undelegateFromAddress(getPrimaryDelegate(msg.sender));  
+}
```

```
368 +uint256 _amount = _balances[msg.sender];
```

```
390 +if (!isOwnDelegate(delegate)) {  
+    _undelegateFromAddress(delegate, getPrimaryDelegate(delegate)  
+        );  
+}
```

```
396 +uint256 _amount = _balances[delegate];
```



## CVF-2.23. FIXED

- **Category** Unclear behavior
- **Source** VoteCheckpoints.sol

**Description** This will revert in case there is no primary delegate.

**Recommendation** Even if this is a desired behavior, Consider handling this case explicitly.

```
365 +undelegateFromAddress(getPrimaryDelegate(msg.sender));
```

```
391 +_undelegateFromAddress(delegator, getPrimaryDelegate(delegator));
```

## CVF-2.24. FIXED

- **Category** Suboptimal
- **Source** VoteCheckpoints.sol

**Description** The code below does make sense only when “\_sourcePrimaryDelegate” is not zero.

**Recommendation** Consider adding an explicit “require” statement to ensure this.

**Client Comment** *The require of ‘amount <= \_undelegatedAmount + \_sourcePrimaryDelegate’ will always fail if \_sourcePrimaryDelegate = 0 since the previous if statement is ‘\_undelegatedAmount < amount’.*

```
567 +address _sourcePrimaryDelegate = getPrimaryDelegate(from);
```

## CVF-2.25. FIXED

- **Category** Flaw
- **Source** VoteCheckpoints.sol

**Description** This ignores “op” and will work incorrectly in case “op” is “subtract”.

**Recommendation** Consider storing “op(0, delta} instead of just “delta”.

```
663 +value: uint224(delta)
```



## CVF-2.26. FIXED

- **Category** Unclear behavior
- **Source** InflationRootHashProposal.sol

**Description** A successful missing account claim is not rewarded.

490    `function claimMissingAccount()`

## CVF-2.27. FIXED

- **Category** Unclear behavior
- **Source** InflationRootHashProposal.sol

**Description** A zero address may have some voting power, thus it cannot be used as a marker of a not challenged index.

**Recommendation** Consider using a separate boolean field as a marker. Even if some code in other smart contract guarantees a zero address to never have any voting power, it would be better not to rely on this.

**Client Comment** *Using the balance field as balance is not allowed to be zero. Also disallowed responding with address(0).*

648    `+if (leftNeighborChallenge.account != address(0)) {`

690    `+rightNeighborChallenge.account != address(0)`

## CVF-2.28. FIXED

- **Category** Suboptimal
- **Source** InflationRootHashProposal.sol

**Recommendation** Should be “>=”, otherwise the tree is still non-deterministic.

939    `+2**(_proof.length - 1) > _numAccounts ||`



## CVF-2.29. FIXED

- **Category** Documentation
- **Source** PolicyVotes.sol

**Description** The comment says that more than 50% of the total voting power is required to early execute a voting, actually, the code accepts a vote when at least 50% of the total voting power or of the total stake voted yes.

**Recommendation** Consider changing the code to require strictly more than 50% yes votes.

**Client Comment** *Comment is wrong, code is right, changed to say "at least".*

271 `+* or after the point that more than 50% of the total voting power`

## CVF-2.30. FIXED

- **Category** Unclear behavior
- **Source** PolicyVotes.sol

**Description** As division rounds down, the calculated required stake could be less than 50% of the total stake.

**Recommendation** Consider rounding up.

279 `uint256 _requiredStake = totalStake / 2;`

## CVF-2.31. FIXED

- **Category** Unclear behavior
- **Source** PolicyVotes.sol

**Description** As division rounds down, the required yes state amount could be less than 50% of the total voting power, while comment explicitly says that more than 50% of the total voting power is required for early execution.

286 `+if (yesStake < _total / 2) {`



## CVF-2.32. FIXED

- **Category** Unclear behavior
- **Source** IsPrime.sol

**Description** What was `_n - 4` is not `_n - 3`.

**Recommendation** Consider checking, which one is correct.

**Client Comment** *n-3 is correct, allows the pseudorandom value to be in the range  $1 < a < n - 1$*

57   +**uint256** \_n3 = \_n - 3;

101   -   **uint256** a = (**uint256**(hash) % (\_n - 4)) + 2;

103   +   **uint256** a = (**uint256**(hash) % \_n3) + 2;

## CVF-2.33. INFO

- **Category** Flaw
- **Source** BigNumber.sol

**Description** This check is incorrect, as the number of significant bytes in a big number could actually be less than `_length * 32` due to padding.

**Recommendation** Consider fixing the check.

**Client Comment** *As `_size` is forced to be a multiple of 32, the scenario you describe would involve a leading word containing all zeros. Our methods all remove those kind of empty words when calculating a result, so I'm not sure how you'd manage to craft that kind of result.*

186   +**require**(`_size` >= `length` \* 32, "Number too large to represent");



## CVF-2.34. INFO

- **Category** Unclear behavior
- **Source** PolicyProposals.sol

**Description** This condition allows refunding the selected but not yet deployed proposal after the period is over.

**Client Comment** *Selecting a proposal is to be done before the time period is over and deployment is to happen immediately, so a situation where a proposal gets selected, but then refunded would require someone supporting at the last second. Without their action, selection would not occur (and therefore no voting), with thier actions voting cannot occur. Therefore exploiting this has no effect on the system and costs the attacker.*

532   **require(**

534   +    proposalSelected && **address**(proposalToConfigure) == **address**(0)  
      **↳ ) ||**  
  +    getTime() > proposalEnds,  
      "Refunds may not be distributed until the period is over"  
  );

## CVF-2.35. INFO

- **Category** Unclear behavior
- **Source** PolicyProposals.sol

**Description** This function is callable by anyone.

**Recommendation** Consider restricting access to it.

**Client Comment** *This is always called atomically with contract cloning and doesn't allow re-calling*

627   +**function** configure(



# 9 Minor Issues

## CVF-2.36. FIXED

- **Category** Documentation
- **Source** ECO.sol

**Description** This comment looks like a pending TODO.

**Recommendation** Consider either implementing or removing it.

31 `+// need to set a default here`

## CVF-2.37. FIXED

- **Category** Bad datatype
- **Source** ECO.sol

**Description** The value “3” should be a named constant.

116 `+if (uint8(bg.currentStage()) < 3) {`

119 `+if (uint8(bg.currentStage()) == 3) {`

## CVF-2.38. FIXED

- **Category** Bad naming
- **Source** ECOx.sol

**Description** The name is confusing. One could think that this value is a denominator and the precision is 2 decimals rather than 100 bits.

**Recommendation** Consider renaming to “PRECISION\_BITS”.

35 `uint8 public constant PRECISION = 100;`



## CVF-2.39. FIXED

- **Category** Bad datatype
- **Source** ECOx.sol

**Description** The type of this argument should be “IERO”.

50    +**address** \_ecoAddr

## CVF-2.40. FIXED

- **Category** Suboptimal
- **Source** ECOx.sol

**Description** The second part of the check doesn’t make sense, as a “uint256” value may never exceed “type(uint256).max”.

53    +\_initialSupply > 0 && \_initialSupply <= type(**uint256**).max,

## CVF-2.41. INFO

- **Category** Overflow/Underflow
- **Source** ECOx.sol

**Description** Phantom overflow is possible here.

**Recommendation** Consider using a shift+divide function similar to the “muldiv” function.

**Client Comment** *The lifetime supply of the token is very far from needing to be concerned about this issue.*

105    +**uint256** \_preciseRatio = safeLeftShift(\_ecoXValue, PRECISION) /  
+      initialSupply;

## CVF-2.42. FIXED

- **Category** Suboptimal
- **Source** ECOx.sol

**Description** This check doesn’t make sense, as a “uint8” value is always less than 256.

116    +**require**(shift < 256, "shift amount too large");



## CVF-2.43. FIXED

- **Category** Unclear behavior
- **Source** DelegatePermit.sol

**Description** This function always succeeds for the zero delegator

**Recommendation** . Consider asserting it is nonzero.

38 +**function** \_verifyDelegatePermit(

## CVF-2.44. FIXED

- **Category** Bad naming
- **Source** DelegatePermit.sol

**Description** Despite the name, this function returns a single nonce, rather than several ones.

**Recommendation** Consider renaming.

72 +**function** delegationNonces(**address** owner) **public view returns** (  
    ↪ **uint256**) {

## CVF-2.45. FIXED

- **Category** Suboptimal
- **Source** EcoXTokenInit.sol

**Description** It would be more efficient to pass a single array of structs with two fields, rather then two parallel arrays. This would also make the length check unnecessary.

21 +**address[]** calldata \_initialHolders,  
+**uint256[]** calldata \_initialBalances



## CVF-2.46. INFO

- **Category** Suboptimal
- **Source** EcoXTokenInit.sol

**Description** This loop doesn't scale.

**Recommendation** Consider implementing an ability to split it across several transactions or allow the initial holders to claim their tokens and pay for gas.

**Client Comment** *\_initialHolders will be 5-6, it just exists for initial decentralization.*

```
31 +for (uint256 i = 0; i < _initialHolders.length; ++i) {
```

## CVF-2.47. FIXED

- **Category** Documentation
- **Source** EcoXTokenInit.sol

**Description** It is a good practice to put a comment into an empty block to explain why the block is empty.

```
36 +constructor() Ownable() {}
```

## CVF-2.48. FIXED

- **Category** Bad naming
- **Source** ERC20Pausable.sol

**Description** The semantics of the returned value is unclear.

**Recommendation** Consider giving a descriptive name to the returned value and/or explaining in the documentation comment.

```
60 +) internal virtual override whenNotPaused returns (uint256) {
```

## CVF-2.49. FIXED

- **Category** Suboptimal
- **Source** ERC20Pausable.sol

**Description** This event is emitted even if nothing actually changed.

```
87 +emit PauserAssignment(_pauser);
```



## CVF-2.50. FIXED

- **Category** Documentation
- **Source** ERC20Permit.sol

**Description** It is a good practice to put a comment into an empty block to explain why the block is empty.

39 +**constructor(string memory name)** EIP712(name, "1") {}

## CVF-2.51. INFO

- **Category** Bad naming
- **Source** ERC20Permit.sol

**Description** Despite the name, this function returns a single nonce rather than several ones.

**Recommendation** Consider renaming.

**Client Comment** *This signature is inherited from the openzeppelin parent contract, so cannot be changed.*

77 +**function** nonces(**address** owner)

## CVF-2.52. FIXED

- **Category** Procedural
- **Source** ERC20.sol

**Description** We didn't review this file.

6 +**import** "../utils/StringPacker.sol";

## CVF-2.53. FIXED

- **Category** Suboptimal
- **Source** EcoTokenInit.sol

**Description** It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays. This would also make the length check unnecessary.

21 +**address[] calldata \_initialHolders,**  
+**uint256[] calldata \_initialBalances**



## CVF-2.54. FIXED

- **Category** Suboptimal
- **Source** EcoTokenInit.sol

**Description** This loop doesn't scale.

**Recommendation** Consider implementing an ability to split it into several transactions, or allow the initial holders to claim their tokens and pay for gas.

```
31 +for (uint256 i = 0; i < _initialHolders.length; ++i) {
```

## CVF-2.55. FIXED

- **Category** Documentation
- **Source** EcoTokenInit.sol

**Description** It is a good practice to put a comment into an empty block to explain why the block is empty.

```
36 +constructor() Ownable() {}
```

## CVF-2.56. FIXED

- **Category** Documentation
- **Source** PolicyInit.sol

**Description** This parameter was removed

**Recommendation** consider removing the comment as well.

```
28 /* //param _tokenResolvers Identifiers for the token contracts
```

## CVF-2.57. FIXED

- **Category** Documentation
- **Source** PolicyInit.sol

**Description** It is a good practice to put a comment into an empty block to explain why the block is empty.

```
92 +constructor() Ownable() {}
```



## CVF-2.58. FIXED

- **Category** Suboptimal
- **Source** PolicedUtils.sol

**Description** Solidity compiler is smart enough to calculate hashes at compile time.

**Recommendation** Consider replacing the hardcoded values with hash expressions.

```
27 //+ keccak256("EC0")
```

```
29 + 0xe0391e627a5766ef56109c7c98e0542c6e96a116720d7c626119be5b67e1813d  
  ↪ ;
```

```
79 //+ keccak256("EC0xStaking")
```

```
81 + 0x3776fba25fb0e7d0848ec503ec48569754f9d46736d6ace08b6eed818399d8e1  
  ↪ ;
```

## CVF-2.59. INFO

- **Category** Unclear behavior
- **Source** PolicedUtils.sol

**Description** Having a separate role for ECO seems weird.

**Recommendation** Consider just assigning some normal roles to an address controlled by ECO.

**Client Comment** *This is not an address controlled by Eco, this is the address of the ECO token contract.*

```
28 +bytes32 internal constant ID_ECO =  
+ 0xe0391e627a5766ef56109c7c98e0542c6e96a116720d7c626119be5b67e1813d  
  ↪ ;
```



## CVF-2.60. INFO

- **Category** Suboptimal
- **Source** ERC1820Client.sol

**Description** Hardcoding mainnet addresses is a bad practice that makes it harder to test contracts.

**Recommendation** Consider passing the ERC 1820 address as a constructor argument and storing in an immutable variable.

**Client Comment** *The ERC1820 contract is deployed to this address on all chains. This value is a predefined address for a contract that will exist on the chain and can be expected to have this value in any environment where it is present.*

12    `IERC1820Registry(0x1820a4B7618BdE71Dce8cdc73aAB6C95905faD24);`

## CVF-2.61. FIXED

- **Category** Documentation
- **Source** VoteCheckpoints.sol

**Description** It is a good practice to put a comment into an empty block to explain why the block is empty.

117    `+ ) ERC20Pausable(_name, _symbol, admin) {}`

## CVF-2.62. FIXED

- **Category** Suboptimal
- **Source** VoteCheckpoints.sol

**Description** The expression “checkpoints[account].length” is calculated twice.

**Recommendation** Consider calculating once and reusing.

167    `+    checkpoints[account].length <= type(uint32).max,`

170    `+return uint32(checkpoints[account].length);`



## CVF-2.63. FIXED

- **Category** Unclear behavior
- **Source** VoteCheckpoints.sol

**Description** So a user cannot forbid new delegations to him, in case he already has some delegations. This looks odd.

**Recommendation** Consider allowing forbidding new delegations even if some delegations do exist.

```
192 +require(  
+    _balances[msg.sender] == getVotingGons(msg.sender) &&  
+    isOwnDelegate(msg.sender),  
+    "Cannot disable delegation if you have outstanding delegations  
+     ↪ to you"  
+);
```

## CVF-2.64. FIXED

- **Category** Bad naming
- **Source** VoteCheckpoints.sol

**Description** The semantics of the returned value is unclear.

**Recommendation** Consider giving a descriptive name to the returned value and/or explaining in a documentation comment.

```
639 +) internal returns (uint256) {
```



## CVF-2.65. FIXED

- **Category** Bad naming
- **Source** IECO.sol

**Description** it would be more conventional to specify argument names in a function signature rather than in a documentation comment.

```
8  +// address to, uint256 amount
+function mint(address, uint256) external;

11 +// address from, uint256 amount
+function burn(address, uint256) external;

16 +// address owner, uint256 blockNumber
+function getPastVotes(address, uint256) external view returns (
    ↪ uint256);

19 +// uint256 blockNumber
20 +function totalSupplyAt(uint256) external view returns (uint256);
```

## CVF-2.66. FIXED

- **Category** Documentation
- **Source** IECO.sol

**Description** 'It is unclear whether the historical value is returned at the beginning of the specified block, or at the end of it, or at some arbitrary moment within the block. Consider clarifying.

```
16 +// address owner, uint256 blockNumber
+function getPastVotes(address, uint256) external view returns (
    ↪ uint256);

19 +// uint256 blockNumber
20 +function totalSupplyAt(uint256) external view returns (uint256);
```



## CVF-2.67. INFO

- **Category** Procedural
- **Source** RandomInflation.sol

**Recommendation** These variables should be declared as immutable.

**Client Comment** *This needs to be mutable for governance's sake so the RandomInflation contract didn't need to be replaced to change this value.*

82 `uint256 public randomVDFDifficulty;`

## CVF-2.68. FIXED

- **Category** Suboptimal
- **Source** RandomInflation.sol

**Recommendation** As long as keys are sequential, it would be more efficient to use a bit mask.

**Client Comment** *We require the ability to go above the amount of tickets trackable by a bitmap, so it will be a combined mapping(bitmap.*

92 `mapping(uint256 => bool) public claimed;`

## CVF-2.69. FIXED

- **Category** Suboptimal
- **Source** RandomInflation.sol

**Recommendation** These parameters should be indexed.

115 `+VDFVerifier vdfVerifier,  
+InflationRootHashProposal inflationRootHashProposal,`



## CVF-2.70. FIXED

- **Category** Flaw
- **Source** RandomInflation.sol

**Description** There is no range check for this argument.

**Recommendation** Consider adding an appropriate check.

```
140 +uint256 _randomDifficulty,
```

## CVF-2.71. FIXED

- **Category** Suboptimal
- **Source** RandomInflation.sol

**Recommendation** This condition could be simplified to "seed != 0" because of the "require" statement above.

```
166 +if (seed != 0 || getTime() < claimPeriodStarts + CLAIM_PERIOD) {
```

## CVF-2.72. FIXED

- **Category** Suboptimal
- **Source** RandomInflation.sol

**Recommendation** Consider maintaining a counter of non-claimed rewards to make this loop unnecessary.

```
173 +for (uint256 i = 0; i < numRecipients; ++i) {
```



## CVF-2.73. FIXED

- **Category** Suboptimal

- **Source** RandomInflation.sol

**Description** "% 2" check is missing.

**Client Comment** These are being removed.

```
237 +!(_primal % 3 == 0) &&
+    !(_primal % 5 == 0) &&
+    !(_primal % 7 == 0) &&
240 +    !(_primal % 11 == 0) &&
+    !(_primal % 13 == 0) &&
```

## CVF-2.74. FIXED

- **Category** Suboptimal

- **Source** InflationRootHashProposal.sol

**Description** This array is redundant.

**Recommendation** Just require the root hashes of unused subtrees to be zero.

```
111 +bytes32[34] public ALLOWED_ZERO_DATA = [
```

## CVF-2.75. FIXED

- **Category** Bad datatype

- **Source** InflationRootHashProposal.sol

**Recommendation** The type of the "\_ecoAddr" argument should be "IECO".

```
353 +constructor(Policy _policy, address _ecoAddr) PolicedUtils(_policy)
    ↪ {
```



## CVF-2.76. FIXED

- **Category** Suboptimal

- **Source**

InflationRootHashProposal.sol

**Description** Storing constants in the storage is suboptimal.

**Recommendation** Consider defining a function that returns zero hash by level like this:  
function allowedZeroData (uint256 level) internal pure returns (bytes32) { if (level < 32) {  
if (level < 16) { if (level < 8) { if (level < 4) { if (level < 2) { if (level < 1) return /\* value #0 \*/;  
else return /\* value #1 \*/; } else { if (level < 3) return /\* value #2 \*/; else return /\* value #3 \*/; } } } } } ....

**Client Comment** Went with the approach detailed in CVF-9 instead.

375 +ALLOWED\_ZERO\_DATA = InflationRootHashProposal(\_self).  
    ↳ allowedZeroData();

## CVF-2.77. FIXED

- **Category** Suboptimal

- **Source**

InflationRootHashProposal.sol

**Recommendation** Shift would be more efficient than power.

939 +2\*\*(\_proof.length - 1) > \_numAccounts ||  
940 +2\*\*(\_proof.length) < \_numAccounts

## CVF-2.78. FIXED

- **Category** Suboptimal

- **Source**

InflationRootHashProposal.sol

**Recommendation** This loop should be executed only if computedHash == \_root.

969 +for (uint256 i = \_proof.length; i > 0; i--) {



## CVF-2.79. FIXED

- **Category** Suboptimal
- **Source** InflationRootHashProposal.sol

**Recommendation** This check could be optimized by pre-calculating `_index ^ (_numAccounts - 1)`.

```
973 +(_index >> (i - 1)) & 1 == ((_numAccounts - 1) >> (i - 1)) & 1
```

## CVF-2.80. FIXED

- **Category** Suboptimal
- **Source** InflationRootHashProposal.sol

**Description** Here a value just written into the storage is read back.

**Recommendation** Consider reusing the written value.

```
1005 +challenge.challengeEnds += CONTESTING_TIME;
```

```
1009 +uint256 challengeEnds = challenge.challengeEnds;
```

## CVF-2.81. FIXED

- **Category** Bad naming
- **Source** TimedPolicies.sol

**Description** The name is confusing, as one could think that this is the duration or any generation, while this is just the minimum curation of a generation.

**Recommendation** Consider renaming to "MIN\_GENERATION\_DURATION".

```
26 uint256 public constant GENERATION_DURATION = 14 days;
```



## CVF-2.82. FIXED

- **Category** Bad naming
- **Source** TimedPolicies.sol

**Description** The name is confusing. As one could think that this is the exact start time of the next generation, while this is just the minimum start time for the next generation.

**Recommendation** Consider renaming.

```
34 uint256 public nextGenerationStart;
```

## CVF-2.83. FIXED

- **Category** Suboptimal
- **Source** TimedPolicies.sol

**Recommendation** This code could be simplified as: notificationHashes = TimedPolicies(\_self).getNotificationHashes();

```
100 +bytes32[] memory hashes = TimedPolicies(_self).  
    ↪ getNotificationHashes();  
+for (uint256 i = 0; i < hashes.length; ++i) {  
+    notificationHashes.push(hashes[i]);
```

## CVF-2.84. FIXED

- **Category** Suboptimal
- **Source** TrustedNodes.sol

**Description** This constant duplicates the "GENERATION\_DURATION" constant defined in the "TimedPolicies.sol" file.

**Recommendation** Consider defining in one place.

**Client Comment** Constant now defined in PolicedUtils.sol as all the relevant contracts inherit from there.

```
20 +uint256 private constant GENERATION = 14 days;
```



## CVF-2.85. FIXED

- **Category** Unclear behavior
- **Source** TrustedNodes.sol

**Recommendation** Should it be “26 \* GENERATION”?

21    +**uint256 private constant** YEAR = 26 \* 14 **days**;

## CVF-2.86. FIXED

- **Category** Bad datatype
- **Source** TrustedNodes.sol

**Recommendation** The initial generation ID should be a named constant.

129    +yearStartGen = 1001;

## CVF-2.87. FIXED

- **Category** Readability
- **Source** TrustedNodes.sol

**Recommendation** Brackets are redundant here.

134    +unallocatedRewardsCount = (\_numTrustees \* YEAR) / GENERATION;

## CVF-2.88. FIXED

- **Category** Suboptimal
- **Source** TrustedNodes.sol

**Description** The value “lastYearVotingRecord[msg.sender]” is read from the storage twice.

**Recommendation** Consider reading once and reusing.

224    +**uint256 record** = lastYearVotingRecord[**msg.sender**];

230    +lastYearVotingRecord[**msg.sender**] -= rewardsToRedeem;



## CVF-2.89. FIXED

- **Category** Documentation
- **Source** PolicyVotes.sol

**Description** This phrase is incomplete.

86    `+/** Event emitted when split vote is.`

## CVF-2.90. FIXED

- **Category** Suboptimal
- **Source** PolicyVotes.sol

**Description** The “PolicySplitVoteCast” event is supersedes” the “PolicyVote” event.

**Recommendation** Consider merging these two events into a single event.

89    `+event PolicySplitVoteCast(`

## CVF-2.91. FIXED

- **Category** Suboptimal
- **Source** VDFVerifier.sol

**Recommendation** Just doing “cmp” would be more efficient as “cmp” internally anyway starts with comparing the numbers lengths.

168    `+ulen > 32 || (ulen == 32 && BigNumber.cmp(u, one) == 1),`

## CVF-2.92. FIXED

- **Category** Suboptimal
- **Source** BigNumber.sol

**Recommendation** This expression could be simplified and optimized as: `(length + 31) » 5`

71    `+uint256 numSlots = (length - 1) / 32 + 1;`



## CVF-2.93. FIXED

- **Category** Suboptimal
- **Source** BigNumber.sol

**Recommendation** Bitwise "AND" would be more efficient.

```
74 +uint256 offset = length % 32;
```

## CVF-2.94. FIXED

- **Category** Suboptimal
- **Source** BigNumber.sol

**Description** This loop is suboptimal.

**Recommendation** Consider using the identity precompile instead.

```
119 +for (uint256 i = 1; i < numSlots; i++) {
```

## CVF-2.95. FIXED

- **Category** Suboptimal
- **Source** BigNumber.sol

**Recommendation** Shift would be more efficient.

```
186 +require(_size >= length * 32, "Number too large to represent");
```

## CVF-2.96. FIXED

- **Category** Suboptimal
- **Source** BigNumber.sol

**Description** The expression `_size / 32 - length` is calculated on every loop iteration.

**Recommendation** Consider calculating once before the loop.

```
200 +for (uint256 i = 0; i < _size / 32 - length; i++) {
```



## CVF-2.97. FIXED

- **Category** Suboptimal
- **Source** BigNumber.sol

**Description** Padding byte by byte is suboptimal.

**Recommendation** Consider calculating the padding size and just skipping this number of bytes before copying the number into the resulting array.

```
200 +for (uint256 i = 0; i < _size / 32 - length; i++) {
```

## CVF-2.98. FIXED

- **Category** Suboptimal
- **Source** BigNumber.sol

**Description** Copying a number word by word is suboptimal.

**Recommendation** Consider using the identity (0x4) precompile or abi.encodePacked instead.

```
205 +for (uint256 i = 0; i < length; i++) {
```

## CVF-2.99. FIXED

- **Category** Suboptimal
- **Source** BigNumber.sol

**Description** Skipping zero bytes one by one is suboptimal.

**Recommendation** Consider using the bisection method like this: if (firstWord » 128 == 0) { firstWord «= 128; offset += 16; } if (firstWord » 192 == 0) { firstWord «= 64; offset += 8; } if (firstWord » 224 == 0) { firstWord «= 32; offset += 4; } if (firstWord » 240 == 0) { firstWord «= 16; offset += 2; } if (firstWord » 248 == 0) { firstWord «= 8; offset += 1; }

```
239 +firstWord > 0 && firstWord & bytes32(uint256(0xff << 248)) == 0
```



## CVF-2.100. FIXED

- **Category** Procedural
- **Source** BigNumber.sol

**Description** Doing the "firstWord > 0" check on every loop iteration is redundant.

**Recommendation** Consider checking once before the loop.

```
239 +firstWord > 0 && firstWord & bytes32(uint256(0xff << 248)) == 0
```

## CVF-2.101. FIXED

- **Category** Suboptimal
- **Source** BigNumber.sol

**Recommendation** Shift would be more efficient.

```
270 +return _base.value.length * 32;
```

## CVF-2.102. FIXED

- **Category** Suboptimal
- **Source** BigNumber.sol

**Description** Here, the number is converted to bytes array just to know the length of this array. This is suboptimal.

**Recommendation** Consider just calculating the length without converting the number to a bytes array.

```
284 +return asBytes(_base).length;
```



## CVF-2.103. FIXED

- **Category** Suboptimal
- **Source** BigNumber.sol

**Description** Doing this check on every loop iteration is suboptimal.

**Recommendation** Consider splitting the loop into two loops. Once from zero to min\_length - 1 and another from min\_length to max\_length.

478 +**switch** gt(i, sub(max\_len, min\_len))

652 +**switch** gt(i, len\_diff)

## CVF-2.104. FIXED

- **Category** Suboptimal
- **Source** BigNumber.sol

**Description** Calculating max\_length - min\_length on every loop iteration is supoptimal.

**Recommendation** Consider calculating once before the loop.

478 +**switch** gt(i, sub(max\_len, min\_len))

## CVF-2.105. INFO

- **Category** Suboptimal
- **Source** BigNumber.sol

**Recommendation** It would be more efficient to compare and calculate the difference in once loop. The current implementation calculates the difference for the initial equal words skipped by the "cmp" function.

**Client Comment** I am comfortable leaving this as-is.

578 compare = cmp(\_a, \_b);

582 + instance.value = innerDiff(\_a.value, \_b.value);

587 + instance.value = innerDiff(\_b.value, \_a.value);



## CVF-2.106. FIXED

- **Category** Suboptimal
- **Source** BigNumber.sol

**Recommendation** It would be more efficient to maintain a pointer to the highest no-zero word inside the result, and update this point when writing the result words. Such approach would make this loop unnecessary.

```
716 } iszero(mload(result_ptr)) {  
    result_ptr := add(result_ptr, 0x20)
```

## CVF-2.107. FIXED

- **Category** Suboptimal
- **Source** BigNumber.sol

**Recommendation** 64 bytes here seems redundant. 1 byte would be enough.

```
811 _modulus = new bytes(64);
```

## CVF-2.108. FIXED

- **Category** Suboptimal
- **Source** BigNumber.sol

**Description** This leaves empty the first 31 bytes of the modulus, which is waste of memory.

**Recommendation** Consider storing 0x8 in the first modulus byte.

```
819 +mstore(add(_modulus, 0x20), 0x1)
```

## CVF-2.109. FIXED

- **Category** Suboptimal
- **Source** BigNumber.sol

**Recommendation** This is redundant. Free memory is anyway not guaranteed to be zero, and the code doesn't write nor care about the modulus bytes after the first significant byte. Thus no need to reserve all the modulus memory.

```
821 +mstore(0x40, add(_modulus, add(modIndex, 0x20)))
```



## CVF-2.110. FIXED

- **Category** Procedural
- **Source** BigNumber.sol

**Description** The "resultPtr" and "length" variables are updated on every loop iteration.

**Recommendation** Consider just calculating the number of leading zeros in the loop. and update the variables once after the loop.

960 `+resultPtr := add(resultPtr, 0x20)`

962 `+length := sub(length, 0x20)`

## CVF-2.111. FIXED

- **Category** Documentation
- **Source** BigNumber.sol

**Description** It is unclear what this comment is about. The function visibility is actually "private".

977 `+// function visibility is changed to internal to reflect this.`

## CVF-2.112. FIXED

- **Category** Suboptimal
- **Source** BigNumber.sol

**Description** This check makes the "\_value" argument redundant.

1005 `require(_value == 0x2, "May only shift by 0x2");`



## CVF-2.113. FIXED

- **Category** Suboptimal
- **Source** CurrencyGovernance.sol

**Description** Sorting on-chain is expensive.

**Recommendation** Consider including into a proposal a presorted array of vote addresses and an array of scores. In this case it would be enough to check that the votes array is sorted ( $O(n)$ ) and that scores are unique. The latter could be done efficiently via a bit mask.

280   +**for** (**uint256** i = 1; i < numVotes; ++i) {  
+     **for** (**uint256** j = i; j > 0; --j) {

## CVF-2.114. FIXED

- **Category** Suboptimal
- **Source** PolicyProposals.sol

**Recommendation** It would be more efficient to merge thses two mappings into a single mapping whose keys are proposals and values are structs encapsulating the values of the original mappings.

64   +**mapping(Proposal => mapping(address => bool)) public** staked;

71   +**mapping(Proposal => Prop) public** proposals;

## CVF-2.115. INFO

- **Category** Suboptimal
- **Source** PolicyProposals.sol

**Recommendation** This variable should be declared as immutable.

**Client Comment** *This needs to be mutable for governance's sake so the RandomInflation contract didn't need to be replaced when policyvotes is.*

128   +**PolicyVotes public** policyVotesImpl;



## CVF-2.116. FIXED

- **Category** Documentation
- **Source** PolicyProposals.sol

**Description** The “proposer” parameter is not documented.

**Recommendation** Consider documenting.

137    +**event** Register(**address indexed** proposer, **Proposal indexed**  
      ↳ proposalAddress);

## CVF-2.117. FIXED

- **Category** Documentation
- **Source** PolicyProposals.sol

**Description** The “supporter” parameter is not documented.

**Recommendation** Consider documenting.

146    +**event** Support(**address indexed** supporter, **Proposal indexed**  
      ↳ proposalAddress);

## CVF-2.118. FIXED

- **Category** Documentation
- **Source** PolicyProposals.sol

**Description** This parameter is not documented.

**Recommendation** Consider documenting.

156    +**address indexed** unsupporter,



## CVF-2.119. FIXED

- **Category** Documentation
- **Source** PolicyProposals.sol

**Description** This parameter is not documented.

**Recommendation** Consider documenting.

182 +Proposal **indexed** proposalAddress

## CVF-2.120. FIXED

- **Category** Suboptimal
- **Source** PolicyProposals.sol

**Description** These functions are overcomplicated.

**Recommendation** They could be simplified as: uint256 end = \_page \* \_resultsPerPage; uint256 begin = \_end - \_resultsPerPage; if (end > totalProposals) end = totalProposals; if (begin >= end) return new Proposal[] (0); Proposal [] memory pageProposals = new Proposal [] (end - begin); for (uint256 i = begin, i < end; i++) { pageProposals [i - begin] = allProposals [i]; } return pageProposals;

**Client Comment** *This function is also getting removed.*

239 +**function** getPaginatedProposalAddresses()

275 +**function** getPaginatedProposalData(**uint256** \_page, **uint256**  
    ↪ \_resultsPerPage)

## CVF-2.121. FIXED

- **Category** Suboptimal
- **Source** PolicyProposals.sol

**Recommendation** This condition could be simplified as: if (\_startIndex >= totalProposals) {

249 +**if** (totalProposals == 0 || \_startIndex > totalProposals - 1) {

286 +**if** (totalProposals == 0 || \_startIndex > totalProposals - 1) {



## CVF-2.122. FIXED

- **Category** Documentation
- **Source** PolicyProposals.sol

**Description** The semantics of the returned value is unclear.

**Recommendation** Consider giving a descriptive name to the returned value and/or describing in the documentation comment.

317    +**function** registerProposal(Proposal \_prop) **external returns** (**uint256**  
    ↳ ) {



## CVF-2.123. INFO

- **Category** Suboptimal
- **Source** PolicyProposals.sol

**Description** String error messages are suboptimal.

**Recommendation** Consider using named errors instead.

**Client Comment** Noted, though the amount of rewriting the 239 require messages in the codebase was deemed too high an overhead.

320	+ "The proposal address can't be 0"
325	"Proposals may no longer be registered because the registration ↳ period has ended"
345	+ "The token cost of registration must be approved to ↳ transfer prior to calling registerProposal"
382	"Proposal contract no longer active"
387	"Proposals may no longer be supported because the registration ↳ period has ended"
394	"In order to support a proposal you must stake a non-zero ↳ amount of tokens"
408	+ "You may not stake in support of a proposal twice"
447	+ "Proposal contract no longer active"
449	+ <b>require</b> (!proposalSelected, "A proposal has already been selected");
452	+ "Proposals may no longer be supported because the registration ↳ period has ended"
459	+ "You have not staked this proposal"
473	+ <b>require</b> (totalProposals > 0, "no proposals to delete");
484	+ <b>require</b> (proposalIndex < totalProposals, "proposal does not exist");
499	+ <b>require</b> (proposalSelected, "no proposal has been selected");



## CVF-2.124. FIXED

- **Category** Flaw
- **Source** PolicyProposals.sol

**Description** This looks like a serious issue.

**Recommendation** Consider adding a flag into the “Prop” structure telling whether proposal fee was actually taken or not.

341    `+// note that currently refunds can be claimed for failed proposals  
  ↳ even if fees were not taken`

## CVF-2.125. FIXED

- **Category** Suboptimal
- **Source** PolicyProposals.sol

**Description** The “totalProposals” variable is read twice.

**Recommendation** Consider reading once and reusing.

354    `+totalProposals += 1;`

360    `+return totalProposals - 1;`

## CVF-2.126. FIXED

- **Category** Suboptimal
- **Source** PolicyProposals.sol

**Recommendation** This condition could be rewritten like this to avoid rounding errors: if `(_p.totalStake * SUPPORT_THRESHOLD_DIVISOR > _total * SUPPORT_THRESHOLD) {`

434    `+_p.totalStake >  
+(_total * SUPPORT_THRESHOLD) / SUPPORT_THRESHOLD_DIVISOR`



## CVF-2.127. FIXED

- **Category** Overflow/Underflow
- **Source** PolicyProposals.sol

**Description** Phantom overflow is possible here.

**Recommendation** Consider using the “muldiv” function.

**Client Comment** *This is at odds with the comment above, I am going with the solution presented in CVF-82.*

435 `+(_total * SUPPORT_THRESHOLD) / SUPPORT_THRESHOLD_DIVISOR`

## CVF-2.128. FIXED

- **Category** Procedural
- **Source** PolicyProposals.sol

**Recommendation** The total stake threshold for a proposal could be calculated once when a proposal is created.

**Client Comment** *The total stake threshold is the same for every proposal, so instead a global value will be created. The value cannot be computed on initialization, though, because it needs to access a previous block checkpoint to be consistent, so it will be computed on proposal registration if not already computed.*

435 `+(_total * SUPPORT_THRESHOLD) / SUPPORT_THRESHOLD_DIVISOR`

## CVF-2.129. FIXED

- **Category** Suboptimal
- **Source** PolicyProposals.sol

**Description** This loop doesn't scale.

**Recommendation** Consider adding the proposal index field into the “Prop” structure.

478 `+for (uint256 i = 0; i < totalProposals; i++) {`



## CVF-2.130. FIXED

- **Category** Documentation
- **Source** PolicyProposals.sol

**Description** This comment is not accurate anymore.

**Recommendation** Actually, this function could be called before the period end in case the selected proposal was already deployed.

525    \* on the ballot **for** the voting phase, and can only be called after  
      ↵ the  
      \* period **is** over.

## CVF-2.131. FIXED

- **Category** Suboptimal
- **Source** PolicyProposals.sol

**Description** This function is overcomplicated and its functionality is actually too simple to be extracted as a function.

**Recommendation** Consider removing this function.

597    +**function** \_getPaginationBounds(**uint256** \_page, **uint256**  
      ↵ \_resultsPerPage)

## CVF-2.132. FIXED

- **Category** Suboptimal
- **Source** PolicyProposals.sol

**Recommendation** As zero value for the “totalECOVotingPower” is special and means “not set”, consider explicitly requiring the “\_totalECOxVotingPower” to be non-zero.

**Client Comment** I also put in a small contingency in the contract that calls this function for if this number is actually zero.

628    +**uint256** \_totalECOxVotingPower,



## CVF-2.133. FIXED

- **Category** Suboptimal

- **Source**

ElectCircuitBreaker.propo.sol

**Recommendation** Calls to “abi.encodePacked” here could be replaced with conversions to “bytes” like this: bytes(““ECO”“)

```
60 +bytes32 _ecoId = keccak256(abi.encodePacked("EC0"));
+bytes32 _ecoXId = keccak256(abi.encodePacked("EC0x"));

65 +bytes32 _currencyTimerId = keccak256(abi.encodePacked(
    ↪ CurrencyTimer));
```

## CVF-2.134. FIXED

- **Category** Bad datatype

- **Source** ECOxStaking.sol

**Recommendation** The type of the “\_ecoXAddr” argument should be “IERC20”.

```
32 +constructor(Policy _policy, address _ecoXAddr)
```

## CVF-2.135. FIXED

- **Category** Documentation

- **Source** StringPacker.sol

**Recommendation** Consider adding some documentation into this contract to describe the packed string format.

**Client Comment** Added documentation, but note that you are the source of this contract:  
<https://medium.com/coinmonks/imutable-string-variables-2a35fc385a41>

```
5 +library StringPacker {
```



## CVF-2.136. INFO

- **Category** Readability
- **Source** ForwardTarget.sol

**Recommendation** Consider using a hash expression instead of a hardcoded hash.

**Client Comment** Only direct number constants and references to such constants are supported by inline assembly.

12 `+// keccak256(abi.encodePacked("com.eco.ForwardProxy.target"))`

14 `+0xf86c915dad5894faca0dfa067c58fdf4307406d255ed0a65db394f82b77f53d4;`

## CVF-2.137. INFO

- **Category** Readability
- **Source** ForwardProxy.sol

**Recommendation** Consider using a hash expression instead of a hardcoded hash.

**Client Comment** Only direct number constants and references to such constants are supported by inline assembly.

13 `+// keccak256(abi.encodePacked("com.eco.ForwardProxy.target"))`

15 `+0xf86c915dad5894faca0dfa067c58fdf4307406d255ed0a65db394f82b77f53d4;`

## CVF-2.138. FIXED

- **Category** Documentation
- **Source** IGeneration.sol

**Recommendation** Should be "@return".

6 `+// returns uint256 generation number`



## CVF-2.139. FIXED

- **Category** Bad datatype
- **Source** IGeneration.sol

**Recommendation** The value “1000” should be a named constant defined in this interface.

**Client Comment** *The GENERATION\_START constant lives in PolicedUtils, I put a comment referring someone to there as the identifier cannot be defined in both.*

7    `+// generations index from 1000`

## CVF-2.140. FIXED

- **Category** Bad datatype
- **Source** EcoBootstrap.sol

**Description** Replacing “uint256” with “uint8” don’t increase performance, and may even decrease it.

**Recommendation** Consider using “uint256”.

24    `-uint256 internal constant NUM_PLACEHOLDERS = 20;  
+uint8 public immutable NUM_PLACEHOLDERS;`

## CVF-2.141. FIXED

- **Category** Bad naming
- **Source** ILockups.sol

**Description** Despite the name this function returns a single lockup, not several ones.

**Recommendation** Consider renaming.

**Client Comment** *Interface was unused and deleted.*

11    `+function lockups(uint256) external view returns (Lockup);`





# ABDK Consulting

## About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

## Contact

### Email

[dmitry@abdkconsulting.com](mailto:dmitry@abdkconsulting.com)

### Website

[abdk.consulting](http://abdk.consulting)

### Twitter

[twitter.com/ABDKconsulting](https://twitter.com/ABDKconsulting)

### LinkedIn

[linkedin.com/company/abdk-consulting](https://linkedin.com/company/abdk-consulting)